Neural Networks, Vol. 1, pp. 339–356, 1988
Printed in the USA.

0893-6080/88 $3.00 + .00

ORIGINAL CONTRIBUTION

Generalization of Backpropagation with Application to a Recurrent Gas Market Model

PAUL J. WERBOS

U.S. Department of Energy

(*Received August* 1987; *revised and accepted May* 1988)

Abstract—*Backpropagation is often viewed as a method for adapting artificial neural networks to classify patterns. Based on parts of the book by Rumelhart and colleagues, many authors equate backpropagation with the generalized delta rule applied to fully-connected feedforward networks. This paper will summarize a more general formulation of backpropagation, developed in 1974, which does more justice to the roots of the method in numerical analysis and statistics, and also does more justice to creative approaches expressed by neural modelers in the past year or two. It will discuss applications of backpropagation to forecasting over time (where errors have been halved by using methods other than least squares), to optimization, to sensitivity analysis, and to brain research.*

This paper will go on to derive a generalization of backpropagation to recurrent systems (which input their own output), such as hybrids of perceptron-style networks and Grossberg/Hopfield networks. Unlike the proposal of Rumelhart, Hinton, and Williams, this generalization does not require the storage of intermediate iterations to deal with continuous recurrence. This generalization was applied in 1981 to a model of natural gas markets, where it located sources of forecast uncertainty related to the use of least squares to estimate the model parameters in the first place.

Keywords—Backpropagation, Recurrent, Continuous time, Reinforcement learning, Energy models, Prediction, Modelling, Cerebral cortex.

1. INTRODUCTION

Backpropagation, as formulated by Rumelhart, Hinton, and Williams (1986) with acknowledgement of the prior work by David Parker (1985), may well be the most widely-used method to adapt artificial neural networks, for use in pattern classification. Nevertheless, the limitations of that formulation have been severely criticized by neuropsychologists and by classical computer scientists. The neuropsychologists have argued that simple feedforward networks cannot do justice to the structure and power of the brain. Neuropsychologists and computer scientists have argued that complex, interesting problems tend to require iterative procedures (or networks) for their solution. Many other criticisms have been raised, which merit serious attention.

Section 2 of this paper will review a different formulation of backpropagation, developed in the period between 1968 and 1974, which can overcome many of these difficulties. This formulation deals with the general case of nonlinear systems of equations. It lacks the concrete, specialized appeal of Rumelhart's discussion, but it can apply to neural networks, econometric models, and other systems as special cases. Applications to prediction, optimization and sensitivity analysis become possible; as an example, this paper will discuss an application to the sensitivity analysis of a natural gas market model developed by the Department of Energy. Werbos (1987a) discussed at length a research strategy for brain research and factory automation based upon this formulation.

Section 3 of this paper will show how derivatives may also be propagated through recurrent networks (such as those discussed by Grossberg, 1976 and Hopfield and Tank, 1986) *without* the expensive storage of information for each iteration (as required by the approach of Rumelhart et al., 1986). Our approach will require storage, however, to handle true external time lags; the significance of this will be discussed, along with ways to implement this storage and issues related to real-time adaptation. When external time lags are totally absent, our method is closely related to the

---

The views expressed in this paper are those of the author, and do not necessarily reflect those of any component of the Federal government.

Requests for reprints should be sent to Paul J. Werbos, Neuroengineering Program, Room 1134, NSF, 1800 G Street N.W., Washington, DC 20550.

method of Almeida (1987), though slightly more general.

Finally, Section 4 will display a practical application of the methods given in Section 3—an analysis of the properties of a natural gas market model, actually used by the Department of Energy several years ago. The conclusions of this analysis were double-checked by explicit numerical perturbations of the model. This analysis provided an insight into the limitations of the model, which are related to certain limitations of multiple regression, the method used to estimate (adapt) the model in the first place. Multiple regression is closely related to the generalized delta rule for network adaptation; however, alternative estimation (adaptation) rules exist which have overcome these limitations in simulation studies and in several practical examples (Werbos, 1974, 1983a, 1988a; Werbos & Titus, 1978). Those alternative rules are consistent with the general framework proposed here.

2. GENERAL FRAMEWORK: BACKGROUND, TERMINOLOGY, AND APPLICATIONS

Rumelhart, Hinton, and Williams (RHW)

Debates about backpropagation have been confused, in part, by different definitions of the word. The index to Rumelhart et al. (1986) defines the word backpropagation by pointing to three pages of text which discuss the generalized delta rule. The generalized delta rule, in turn, is defined as a set of three steps to be applied to feedforward networks. (RHW also discuss recurrent networks, but that extension will not be discussed until Section 3, in order to simplify things here.) RHW specify feedforward networks as:

$$o_{pj} = f_j(\text{net}_{pj}) \tag{1}$$

$$\text{net}_{pj} = \sum_k w_{jk} o_{pk} = \sum_k w_{jk} f_k(\text{net}_{pk}), \tag{2}$$

where o_{pj} is the output of unit number j for pattern (observation) number p, where f_j is some differentiable function, where w_{jk} is a weight to be adapted, and where the processing units are assumed to be ordered in a feedforward fashion. In a feedforward network, the summation over k in Equation (2) can run from 1 to $j-1$, in principle. Some readers have interpreted this to mean that the network must be fully connected; however, even in the RHW formulation, most of the w_{jk} could be fixed to zero, in a practical application, so that the physical connections and the required calculations can both be sparse.

The first of the three steps in the generalized delta rule (p. 327) is a calculation for the final outputs of the network:

$$\delta_{pj} = (t_{pj} - o_{pj}) f'_j(\text{net}_{pj}), \tag{3}$$

where t_{pj} is the teaching or target value for the output of unit number j and f'_j is just the derivative of f_j. This step is explained (p. 323) by noting that δ_{pj} is just the derivative of error, E_p, with respect to net_{pj}, defined as:

$$E_p = \frac{1}{2} \sum_j (t_{pj} - f_j(\text{net}_{pj}))^2 \tag{4}$$

The second step (p. 327) is a calculation for all other units j which output to units k:

$$\delta_{pj} = f'_j(\text{net}_{pj}) \sum_k \delta_{pk} w_{kj}$$

$$= \sum_k \delta_{pk} f'_j(\text{net}_{pj}) w_{kj} \tag{5}$$

This step is explained (p. 326) as a way of calculating the derivatives of E_p with respect to all of the net_{pj}, in a single pass of calculations, based on an informal appeal to the chain rule for differentiation. The third step (p. 330) is a procedure to adapt the weights of the network:

$$\Delta w_{ji}(n+1) = \alpha(\delta_{pj} o_{pi}) + \eta \Delta w_{ji}(n) \tag{6}$$

Note that $\delta_{pj} o_{pi}$ is simply the derivative of E_p with respect to w_{ji}.

A More General View of Backpropagation

Researchers in this field sometimes use the term backpropagation to refer to the second step above, or to all three steps, with or without variations. Again, Rumelhart et al. (1986, Index) appear to refer to all three steps. We would propose that the term backpropagation should include any three-step or three-component procedure for adapting a network, in which the three steps are:

- An output evaluation component (OEC), which evaluates how successful the ultimate outputs of the network are in minimizing or maximizing something. In other words, the OEC defines what the network is supposed to minimize or maximize. More precisely, the OEC provides the derivatives of some evaluation function (such as error) with respect to the ultimate outputs of the network. Equation (3)—the OEC of the generalized delta rule—calculates the derivatives of square error, the error function which is minimized in nonlinear regression; thus, from a statistician's point of view, the generalized delta rule is basically one more numerical way to implement nonlinear regression, a well-known, well-studied statistical method. (See Brode, Werbos, & Dunn, 1975; Dennis & Schnabel, 1983; SAS Institute, 1986; Werbos, 1988a.)
- Dynamic feedback, a method for calculating the derivatives of error or loss with respect to the intermediate outputs and weights *within* the network. (Werbos, 1974, 1982.) Strictly speaking, this is the only component which actually propagates information backwards along a network. This paper will use the term "dynamic feedback" to refer to this

component, in part because this term was used in the original papers on this concept, and in part because the term backpropagation usually refers to the combination of all three components.

- A convergence method or solution algorithm, a method for *responding* to the derivatives (and/or other local information) by adapting the parameters. This may involve a simple proportionate response (steepest descent), or conjugate gradient methods (which include (6) as a special case, but which provide procedures for adjusting the sensitivity constants α and η), or more complex methods like those which have worked in complex practical applications (Werbos, 1983a). Surprisingly, some of the classical methods from statistics and numerical analysis (Dennis & Schnabel, 1983) can be applied with $O(n)$ storage in exact or near-exact form (Werbos, 1988b).

Origin of the General View

Background. The intuitive notion of backpropagation—of adaptation and optimization based on a flow of feedback backwards through a neural system, specifically related to the issue of brain functioning and artificial intelligence—was published in Werbos (1968), albeit in a clumsy linear version. A nonlinear version, essentially equivalent to the generalized delta rule, was proposed in various documents circulated in 1971 and 1972. At that time, applications to artificial neural networks were not considered interesting or acceptable to much of the scientific community. Therefore, the method was generalized to permit applications to more conventional forecasting applications (Werbos, 1974).

Werbos (1974) also cited related work in control theory, which also used backwards flows of information to identify systems, albeit in a different way. The formulation to be given below could have been derived as an extension of control theory, but I found it easier simply to prove (9) directly. Likewise, I found it much easier to apply (9) directly to neural-like problems than to extend and generalize the more complex and indirect methods of control theory. This is especially true with stochastic optimization, where the notation can otherwise get quite complex. Nevertheless, a reviewer has suggested that Athans and Kalb (1966) came surprisingly close to the kind of approach presented here; the details are beyond the scope of this paper, in part because I have never seen the book. For an easy tutorial on my 1974 formulation of backpropagation and various alternatives, see Werbos (1988a).

The generalized formulation of 1974 began by observing that the "training signal" (t_{pj} in Equation 3) is really just a vector $\mathbf{t}_p$ which the network tries to reproduce or predict. Any set of functional relations can be represented as a network. Likewise, the problem of "adapting weights" in a neural network is just a special case of the problem of estimating the parameters of a general functional model. The use of square error and steepest descent in estimating a model had been established decades before; therefore, the novel feature of backpropagation in this formulation was the use of dynamic feedback in combination with those two components.

(First order) dynamic feedback was defined as a method for calculating the derivatives of some function, L, of the inputs and outputs of a feedforward system, in a single pass through the system.

Feedforward Systems. A feedforward system is defined as follows, in the most general formulation. First, there are m input variables, x_1 through x_m, which include all of the parameters or weights of the system, as well as those variables which are normally thought of as inputs to the system. (By including the weights as variables, one simplifies some of the later calculations.) These variables form an m-component vector, $\mathbf{X}$. Then let x_1 through x_N denote *all* of the variables of the system; these variables form an N-component vector $\mathbf{x}$, of which $\mathbf{X}$ is essentially a subset. Let f_j, for $j = m + 1, \ldots, N$, be the differentiable functions which correspond to the functions implemented by the network components. This means that for $j = m + 1, \ldots, N$:

$$x_j = f_j(x_1, \ldots, x_{j-1}). \tag{7}$$

Finally, we denote the function which we wish to minimize (or simply to differentiate) as:

$$L = L(x_1, \ldots, x_N). \tag{8}$$

Note that this paper will frequently use small letters (like $\mathbf{x}$) to refer to internal inputs or functions *within* a system, and capital letters (like $\mathbf{X}$) to refer to the inputs or outputs of the system *as a whole;* this distinction is important, because both levels of analysis will be discussed.

The network formulation in (7) and (8) is more general than it might appear at first. As with (1) and (2), for example, the functions f_j may form a sparse network, in practice, which simplifies the calculations. To make this apparent, and to make the applications to parallel computers more explicit, I have sometimes spelled out (7) explicitly for the special case of a multilayer network (Werbos, 1987a, Appendix); however, this paper will try to be more general and to avoid the additional notation required to make that example explicit.

Notice that (7) and (8) make no reference to time t or to pattern number p. As a result, there is a choice between two (or more) different ways of using these equations in practice to represent a network. When there is no connection at all between variables at different times or for different patterns (as in Equations 1, 2, and 4), it is possible to identify the variables of the system at any time with the variables x_i of Equations (7) and (8). For example, the RHW system can be

represented in our framework by identifying our series $x_1 \cdots x_N$ with the following RHW variables, in order:

$$w_{jk} \text{ and other inputs, } \text{net}_{p1}, \text{ net}_{p2}, \cdots \text{ net}_{pn},$$

where n is the number of neurons and by identifying our L with their E_p. (Note that the variables o_{pi} for neurons within the system are not necessary; Equations (2) and (4) can represent the system without referring to them.) We can then go on to calculate the derivatives of L with respect to the weights *for each pattern individually,* as RHW do, and then add up these derivatives across different patterns. The details of this equivalence are discussed in Werbos (1988a, 1988b). In brief, the RHW feedforward networks are a special case of equations 7 and 8.

When studying dynamic systems, this kind of simple formulation is not possible. For example, if net_{t1} uses $\text{net}_{t-1,2}$ as one of its inputs, where "t" refers to time and "$t-1$" is the previous observation or pattern, then a more complex use of (7) is needed. Each variable x_i in Equation (7) would then refer to a specific neuron *at* a specific time; the activation level of the same neuron at a different time would have to be treated as a different variable, for purposes of (7). In this case, (7) would say that each neuron is allowed to input the outputs of *earlier neurons* from the *same time,* as well as the outputs of *all neurons* from earlier times. Sections 3 and 4 will give more examples of this sort. In some applications at the Department of Energy, we have even worked with systems where *two* time-dimensions were necessary (Werbos, 1988a); even there, there was no difficulty in using dynamic feedback, because there was a definite sequence of calculation, which determined which variables at which points would be calculated in which order.

The Chain Rule. First-order dynamic feedback is defined as the use of the chain rule for ordered derivatives, in order to calculate the derivatives of L with respect to the system inputs. The chain rule for ordered derivatives (proven in Werbos, 1974) may be written:

$$\frac{\partial^+ L}{\partial x_k} = \frac{\partial L}{\partial x_k} + \sum_{j=k+1}^{N} \frac{\partial^+ L}{\partial x_j} \frac{\partial f_j}{\partial x_k}, \tag{9}$$

where the plus signs indicate ordered derivatives, and the derivatives *without* plus signs refer to conventional partial derivatives of the functions L and f_j. The conventional partial derivatives are calculated by differentiating the functions L and f_j *as they would normally be written,* as functions of their direct arguments as listed in (7) and (8) *without* any substitutions. Since the functions f_j usually depend on only a small portion of the earlier variables, x_k, in practice, the partial derivative on the far right is usually zero for most combinations of j and k; therefore, the summation on the right is usually very sparse and simple. In formal terms, the ordered derivative of L with respect to x_k refers to the derivative of L *expressed as* a function of $x_1 \cdots x_k$, where the dependency of L on $x_{k+1} \cdots x_N$ has been eliminated by substituting in from the equations (7) which equate their values to the functions $f_{k+1} \cdots f_N$. In intuitive terms, the conventional partial derivative refers to the *direct* causal impact of x_k on L, while the ordered derivative refers to the *total* causal impact, including direct and indirect effects, both.

Equation (9) is usually simple to apply as a recursive relation, in practice. One begins by calculating the ordered derivative with respect to x_N, for which the summation on the right is null. One then proceeds backwards to x_{N-1}, x_{N-2}, on down to x_1. For example, to apply (9) to the RHW system (Equations 1, 2, and 4), one would normally begin by allocating an array to hold the ordered derivatives; "delta(k)" could be used to hold the ordered derivative of L with respect to x_k. Then, for each variable x_k in the system, one would identify *which other variables* (x_j) that variable may have a direct impact on; one would differentiate the functions f_j with respect to x_k, and substitute the result into (9), which then becomes a concrete recursion equation for the special case at hand.

In the RHW system, for example, (5) is the special case of (9), where "delta" is used to hold the ordered derivatives, and where x_k is one of the internal variables net_{pj}; this is particularly obvious when we compare the rightmost side of (5) with the conventional derivatives of the rightmost side of (2). (Note that *our* f_j here includes the whole right side, and not just RHW's function "f_k".)

Note that (9) also eliminates the artificial distinction between neurons whose output goes outside the system and neurons whose output is used internally. As with (7), there is no reason to limit oneself to fully connected, rigidly structured networks; if (7) has a sparse structure, which allows for efficient implementation on a parallel computer or circuit, then (9) will automatically have this property as well, at least if it can add efficiently.

Ordered derivatives are important in many other applications besides neural nets. As a result, a host of informal names have been developed for this concept, as used in different applications. For example, economists speak of impact multipliers, control theorists speak of variational derivatives, and many people speak of time-dependent Lagrange multipliers. The use of mathematically oriented language may help reduce the kind of fragmentation which encourages workers in different applications to continually reinvent the wheel.

Early Applications and Development. The first actual application of backpropagation was in estimating time-series models used to predict nationalism and social communications, developed by Prof. Karl Deutsch. Ironically, backpropagation was *not* used to implement ordinary least squares (regression), which had already

been tested in this application by use of conventional software. Instead, it was used to implement more advanced statistical methods, which would have been too costly to use without backpropagation. The results were documented in Werbos (1974), embedded (and documented) in user-oriented software in an MIT version of the Time-Series Processor (Brode et al., 1975), and discussed in Werbos (1977), which emphasized the potential value of the same general mathematics for forecasting and for brain modeling. A general survey of applications—to neural modeling, optimization, sensitivity analysis, and estimation—was presented to the International Federation for Information Processing (IFIP) in 1981 (Werbos, 1982), along with diagrams illustrating both Equation 9 and several generalizations to calculate second-order derivatives economically. These generalizations were quite different from David Parker's second-order backpropagation, which is essentially a new alternative to steepest descent as a convergence method (Parker, 1987). The primary ideas here were widely transmitted, both in writing and otherwise.

Applications of Backpropagation in the General View: Prediction

In the delta rule, the target vector $\mathbf{t}_p$ is a vector to be reproduced or predicted by the network. The distinction between reproduction and prediction is essentially meaningless here, since in both cases we try to match the target vector over previous observations and we hope that the match will still be valid in future observations. Most of the current research on backpropagation—like our own empirical work—has focused on this problem of reproduction or prediction.

Using the notation of statistics, the delta rule is trying to address the well-known problem of estimating $\mathbf{b}$ so as to improve the predictions:

$$\hat{\mathbf{y}}(t) = \mathbf{F}(\mathbf{X}(t), \mathbf{b}), \tag{10}$$

in the special case where $\mathbf{F}$ happens to be represented as a network of elementary units, where the parameters $\mathbf{b}$ happen to be interpreted as a collection of weights, and where the observations (t) may be interpreted as patterns presented to the system. Here, $\mathbf{X}(t)$ is the vector of inputs for observation or time number t, and $\hat{\mathbf{y}}(t)$ is a prediction of the target vector $\mathbf{y}(t)$. As discussed after (7), we use a capital letter (e.g., $\mathbf{F}$) to refer to the vector function which describes the system as a whole; this is different from the f_j, the functions which represent individual components of the system. In a feedforward system, the components of $\hat{\mathbf{y}}(t)$, $\hat{y}_1(t)$ through $\hat{y}_n(t)$, would correspond to the last n components of the vector $\mathbf{x}$ as given in (7).

The use of backpropagation in some form is basically necessary to solve this problem. One cannot find the value of $\mathbf{b}$ which best fits the historical or training data unless one has some definition of the word "best," some measure of the quality of fit; thus an error measure or loss function (such as Equation 4 or the many alternatives used by statisticians) is more or less unavoidable. Admittedly, this measure might not be an explicit part of the adaptation procedure. More importantly, however, one cannot expect to *minimize* such a measure efficiently without exploiting the derivatives of that measure with respect to the parameters. Long experience in numerical analysis has shown the central, unavoidable importance of knowing the derivatives when minimizing or maximizing a complex function of many variables (Dennis & Schnabel, 1983). This makes it essential to use dynamic feedback—to calculate the derivatives at an acceptable cost—to adapt any complex network $\mathbf{F}$, *in the general case;* that in turn leads to the backpropagation strategy.

To improve the power of backpropagation in coping with the prediction problem, one needs to look more closely at each of its three components, and at the formulation of the prediction problem itself. The output evaluation component (like Equation 3) or error function (Equation 4) define what a statistician would call the estimation method. The other two components are simply a numerical procedure for implementing or approximating the estimation method.

Unfortunately, (10) does not do full justice to the kinds of prediction problems which occur in many applications. For example, in econometric forecasting (Werbos, in press), the variables to be predicted are often predicted as functions of their *own values* at a previous times. In other words, the problem is to estimate $\mathbf{b}$ in:

$$\hat{\mathbf{X}}(t+1) = \mathbf{F}(\mathbf{X}(t), \mathbf{u}(t), \mathbf{b}), \tag{11}$$

where $\mathbf{X}$ is a vector of observed variables to be predicted, where $\mathbf{b}$ is a vector of weights or parameters, and where $\mathbf{u}$ is a vector of auxiliary input variables. This kind of prediction over time is also essential when performing optimization over time (to be discussed in the next section). Even Grossberg's explanations of learning require the existence of circuits which somehow learn to produce expectations or predictions of the near-term future (Grossberg, Levine, & Schmajuk, 1987).

Superficially, (10) and (11) may appear to be special cases of each other. For example, we can use a supervised learning system, based on Equation 10, to predict $\mathbf{X}(t+1)$, simply by defining $\mathbf{y}(t)$ as $\mathbf{X}(t+1)$ and defining the system input vector as $\mathbf{X}(t)$ combined with $\mathbf{u}(t)$. Unfortunately, this approach does not lead to the best possible forecasts over time, especially if one is concerned with predictions over more than one period into the future. When ordinary regression (least squares) is used to estimate a model which predicts variables at time $t+1$ as a function of time t, then the forecasts for several months out will tend to deteriorate,

due to cumulative error effects. (There are tricks to avoid this, in some kinds of econometric models, which would not work for neural networks.) Cumulative errors of this sort would be impossible or unavoidable if reality fit a simple model, perfectly, and if all errors were due to random white noise; however, this is not generally the case (Werbos, 1983a, in press). Section 4 will discuss an example of this problem, in detail, as it arises in a real-world forecasting model based on least squares. More to the point, better forecasts have been obtained, in many empirical examples and in simulation studies, by using estimation methods which explicitly represent the notion of forecasting over time (Werbos, 1974, 1983a, in press; Werbos & Titus, 1978). The best results have been obtained with methods which explicitly try to minimize error in multiperiod forecasting.

These methods can be translated into recipes for building neural networks by adopting the 3-net architecture shown in Figure 1. In mathematical terms, Figure 1 represents a 3-equation model used to predict $\mathbf{X}(t)$:

$$\hat{\mathbf{R}}(t+1) = \mathbf{F1}(\mathbf{R}(t), \mathbf{b1}) \quad (12a)$$

$$\mathbf{R}(t+1) = \mathbf{F2}(\hat{\mathbf{R}}(t+1), X(t+1), \mathbf{b2}) \quad (12b)$$

$$\hat{\mathbf{X}}(t+1) = \mathbf{F3}(R(t+1), \mathbf{b3}). \quad (12c)$$

where *all three* functions are implemented as feedforward networks and where some additional arguments would be allowable (Werbos, 1987b). The vector functions **F1** and **F3** both represent networks to predict something, while **F2** calculates what **F1** tries to predict; nevertheless, one can adapt all of the weights together—**b1**, **b2**, and **b3**—by trying to minimize the sum of squared error across all components of **X** and all components of **R**, across time. Dynamic feedback can calculate the derivatives needed in this minimization. See Werbos (1988b) for the details of how to implement this, using RHW-like networks.

This 3-net arrangement has close connections with statistical methods associated with Box and Jenkins and Kalman filtering; for example, in the simplest applications of Equations (12), the **R** vectors would be filtered versions of the **X** variables. Werbos (1987b) elaborates on these connections, and explains how this kind of procedure can lead to more accurate forecasts over time than a direct use of (10) would; furthermore, it explains why it is usually important to minimize a *weighted* sum of squared errors, and to give different weights to different variables. For example, when the prediction networks are used as part of a larger optimization system (to be described), there will automatically be information available about the derivative of long-term utility with respect to each variable R_i and X_i; if variables are weighted according to the variance of these derivatives (plus the variance of error derivatives as well), then the system will give greatest weight to stable, persistent variables—which should lead to better multiperiod forecasting. Note also that the variables R_i, in Equations (12a) and (12b), depend on their own past values, in such a way that the system may "remember" a few periods back; Equation (11) does not provide that capability.

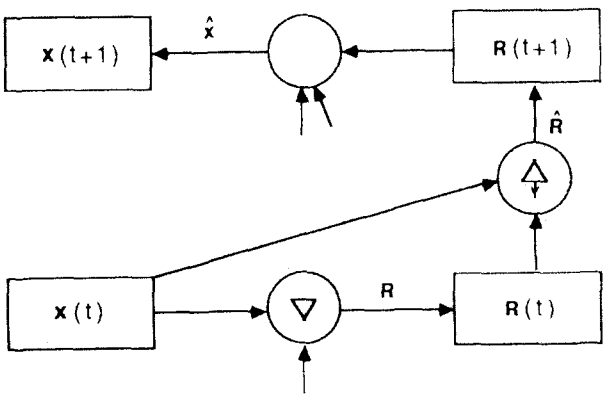

FIGURE 1. Three-net architecture.

Many researchers have criticized the use of backpropagation even with (10), the problem of supervised learning. Some have recommended the use of content-addressable memory instead. Content-addressable memory systems may converge faster than backpropagation, but this would be of little interest if they were converging to the wrong answers (i.e., inconsistent estimators of the weights). Under certain conditions, however (Werbos, 1987b, 1987c), we have found that their estimates may be justified, statistically, if we account for the role of prior probabilities (discussed in Werbos, in press). When these conditions can occur, the ideal adaptation scheme would be a *synthesis* of least squares and content-addressable memory. A synthesis of this sort could be used instead of simple least squares in adapting (12a) and (12c) in the 3-net architecture (though (12b) is a different matter).

In general, these kinds of statistical methods offer a hope of greater robustness, statistical efficiency, and generalizability, based not on speculation but on decades of experience with a huge variety of applications. All of these hopes involve the *accuracy* of the predictions which result when the network is used to predict *new situations*, not in the training set. This still leaves open the questions of how to propagate the required derivatives through a network, and of how to choose a convergence method.

The choice of convergence method (like Equation 6) should not be confused with the choice of estimation method. The convergence method basically determines the *number of iterations* or *cost* of minimizing error over the training set. Admittedly, there are some error functions which place a greater stress on the convergence method, because they are harder to minimize. These are mainly "stiff" error functions, which contain sharp hills and valleys when graphed as a function of the parameters **b**. Smooth, fuzzy error functions are

easier to minimize. Unfortunately, the error or fuzziness in estimating parameters is directly related to the fuzziness of the error function; therefore, those error measures which pinpoint the weights most accurately are precisely those error measures which are hardest to minimize. In summary, one should not expect superior estimation methods (error functions) to reduce the number of iterations required to analyze a fixed training set with a fixed convergence method; one might even expect the opposite. To reduce the number of iterations, we should try instead to develop more powerful convergence methods, which are capable of supporting more sophisticated estimation methods. Fortunately, there are many convergence procedures which have worked on complex practical problems which steepest descent (or its equivalents) could not handle (Dennis & Schnabel, 1983; Werbos, 1983b, 1988b).

When there is no fixed training set (as in organic intelligence, where experience accumulates steadily and old events cannot be truly relived), there are additional complexities; however, we cannot expect to understand these complexities until we understand the simpler situation of adaptation with fixed training sets.

Applications of Backpropagation In the General View: Optimization Over Time

There are many practical problems where a "target vector" would not be available. For example, in robotics, we may know what a robot is supposed to accomplish, but we may not know a priori what its schedule of movements should be to accomplish its task at minimum cost. Instead of a target vector, we may have a notion of what we want the system to accomplish *over time,* a notion which implies some kind of success measure or utility function to be maximized *over time.* If we cannot devise such a measure, then we cannot discriminate between better performance and worse performance, and we cannot say whether our design was successful or not even after the fact. Also, there is no assumption here that the system must have access to an explicit representation of the utility measure as a function (though such information can be exploited, if available).

This problem of utility maximization over time may also be a useful representation of adaptation problems faced by organic systems (Werbos, 1986, 1987a). Hinton (1987) has referred to this problem as the reinforcement learning paradigm. Unlike the paradigm of totally unsupervised learning, it provides an explicit basis for Unconditioned Stimuli or primary reinforcement, which ensures that a system will not be essentially indifferent to biological drives and social feedback.

Werbos (1987a) has shown how this optimization problem can be solved (approximately) by tying together three distinct networks, *each* to be adapted by backpropagation but each with a different output evaluation component. The basic idea is illustrated in Figure 2 (although there are further complexities required to extend the idea to systems as complex as the human brain).

The middle box in Figure 2 basically contains the entire system shown in Figure 1. (Figure 2, like Figure 1, is taken from previous papers using slightly different notation.)

The upper box—the "strategic assessment" network or "J network"—outputs something like an *evaluation* of how well the system is doing, in making progress towards its goals. More precisely, this network would represent an approximation to the "J" function for the optimization problem. The J function comes from dynamic programming, and is defined as follows: the strategy of maximizing J in the short-term (i.e., picking actions $\mathbf{u}(t)$ so as to maximize $J(t+1)$) is equivalent to maximizing the utility function U in the long term (maximizing expected $U(t')$ over all future times t'). Intuitively, the J function corresponds to the notion of conditioned reinforcement, to the static position evaluators sought in game-playing artificial intelligence, to the measures of net present value used by economists, and to other similar ideas (Werbos, 1986).

As with the problem of prediction; there are several different methods which could be used to adapt the J network. One of them—heuristic dynamic programming (HDP) (Werbos, 1977, 1987a)—is similar to conventional backpropagation, with the network adapted to make its output variable, $J(\mathbf{R}(t))$, do a good job of predicting $U(t) + J(\mathbf{R}(t+1) - \bar{U})$, where

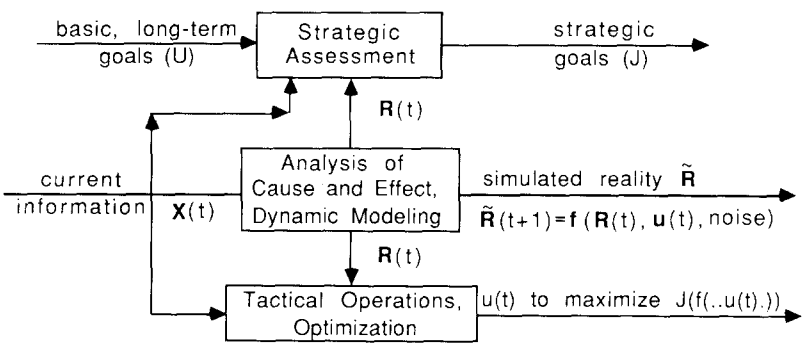

FIGURE 2. Three core components of an intelligent system (J, f, u).

U is the utility measure and $\bar{U}$ is a constant used to prevent drift in the range of the function. More realistically, $U(t)$ may be physically represented as a weighted sum $U_i x_i(t)$, such that the final output of the J network can be a set of components, J_i, each adapted to predict the corresponding $U_i x_i(t) + J_i(t+1) - \bar{U}_i$. Another method—dual heuristic programming— is slightly more sophisticated, but can still be implemented by use of first-order backpropagation. A third method—globalized dual heuristic programming (GDHP)—fully exploits the cause-and-effect information embedded in the middle box, but requires second-order dynamic feedback for its implementation (Werbos, 1987a, 1979, 1982, 1988b); the details are too complex to present here. The required form of second-order dynamic feedback, like first order dynamic feedback, calculates all the required information at a cost which is only proportional to the cost of one pass through the original network.

Using the HDP method, at least, it is possible to forego the middle box and use experience itself (without any simulations) to adapt the J network. The resulting J network would be quite similar to the adaptive position evaluator used in Samuels' checker playing program, or to the adaptive critic used by Barto, Sutton, and Anderson (1983). The work of those authors proves that adaptive optimization is already a practical (or superior) alternative to conventional methods. Sutton has also noted the need for an adaptive model to predict the environment when dealing with more complex problems, like those of robotics in realistic factories. Optimization through backpropagation could also be implemented in more conventional software for use in policy analysis, business decision-making, and the like (Werbos, 1986).

The network in the bottom box would simply determine the actions, $\mathbf{u}(t)$. It would use the derivatives of J (propagated back through the other networks) as its output evaluation component.

3. PROPAGATING DERIVATIVES IN RECURRENT SYSTEMS

Overview

This section will derive a procedure for calculating the derivatives of any evaluation function L with respect to the weights and intermediate variables in a recurrent network. The function L could represent prediction error, or a J function (as defined above), or simply a function we are interested in studying for its own sake. It can be any differentiable function of the network variables (as in Equation 9).

The purpose of this section is simply to generalize the dynamic feedback procedure, for use with networks more general than (7). This generalization could be used with some of the complex architectures from Section 2, but we will deliberately avoid limiting ourselves to those special cases (just as we did in formulating equation 9 itself). In order to visualize this generalization, it may be easier to think in terms of simple supervised learning problems (as in Equation 10) where the function $\mathbf{F}$ can be represented as a recurrent network (not a feedforward network). In other words, this generalization allows one to adapt networks just like (1) and (2), except that all neurons are allowed to input the results of *all other* neurons, without regard to which neuron is earlier and which neuron is later. As the critics of backpropagation have pointed out, a *single layer* network of this kind can represent very complex algorithms which cannot be represented in simple feedforward networks; for example, it could learn to represent the specific, iterative calculations which are fundamental to applications work in adaptive object recognition and speech recognition.

Rumelhart et al. (1986) define recurrent networks as networks in which a unit can take input from units downstream from them, though with a time delay. In describing their basic framework (Rumelhart et al., 1986, Chap. 2), they stress that the time delay is intended to be an approximation to a continuous-time system, the kind of system which Grossberg (1976) and Hopfield and Tank (1986) have written about. They assume that a pattern (p) is presented to the system, and that the experimenter can wait until the state of the network settles down in response to that pattern. Their general framework allows for some relation between a pattern p and earlier patterns, but they admit that the existing work (like their Chap. 8) does not really address that possibility.

Figure 3 illustrates the RHW approach to backpropagation in recurrent networks. For each pattern, the vector $\mathbf{x}$ is allowed to "settle down" for S cycles of the iterative procedure used to approximate a self-consistent state of the network. Backpropagation (Equation 9, in effect) is applied in its usual form by treating variable values in later cycles as distinct variables, downstream from earlier versions of the same variable. (For example, $x_i(p, s+1)$ is treated as a distinct variable, different from and later than $x_i(p, s)$.) To calculate derivatives all the way back to the start ($\mathbf{x}(p, 0)$), it is necessary to work back through all the intermediate values; that, in turn, requires that the intermediate values be stored. For further details, see Rumelhart et al. (1986).

For our purposes, it is extremely important to allow for the interaction between different patterns p, because these patterns may refer to different states in the evolution of the *external* environment across time. I will use the letter "t" (instead of "p") to refer to time in the *external* environment, not in the system per se. Some critics have argued that the brain cannot possibly track discrete time intervals or distinct patterns the way a computer might; however, Purpura (in F. O. Schmitt,

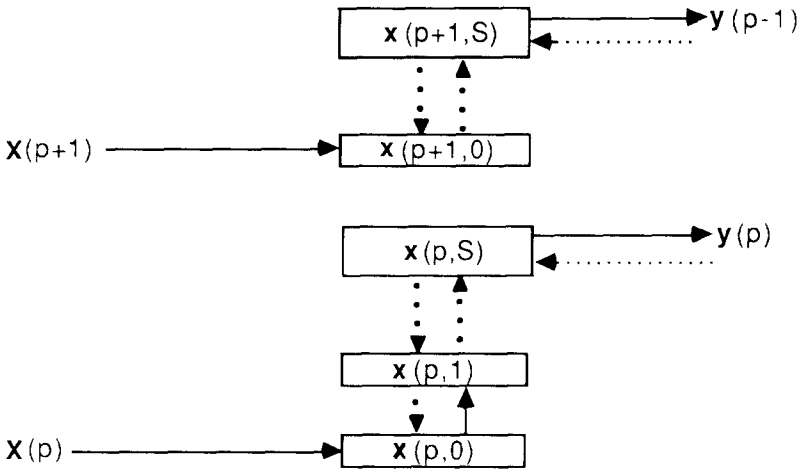

FIGURE 3. RHW approach to recurrent networks.

1970, 1971) has observed discrete clock pulses, of the required sort, going from the nonspecific thalamus to the giant pyramid cells of the cerebral cortex. Foote and Morrison (1987) have observed similar pulses from subthalamic centers.

In our framework (illustrated in Figure 4), there are actually two kinds of recurrence to be considered:

- Time lags, in which the present system output is a function of earlier signals from the previous *external* time period $(t - 1)$.
- Grossberg/Hopfield recurrence, in which there is an immediate response to other units.

Our approach still requires the storage of a complete database, including at least $\mathbf{X}(t)$ and $\mathbf{y}(t - 1)$ for all external time intervals (patterns) t. Such a database is normally built up anyway in standard statistical analysis programs, and we have handled this kind of recurrence in our earliest work (Werbos, 1974). Such a database is not built up in true real-time systems like the brain, to be discussed at the end of this section.

For the second kind of recurrence, we will calculate the required derivatives directly, without using knowledge of intermediate approximations; this is the main difference between the current paper and earlier forms of backpropagation. As in conventional backpropagation through feedforward networks, the cost in time and the cost in storage are both about the same as the costs of running the network in the forwards direction. Two versions of this method will be presented—a version aimed at aggregate-level calculations (see Equations 30, and their application in Section 4), and a version aimed at continuous-time neural networks (Equations 31 and 32, and auxiliary equations).

When the second kind of recurrence is present, but not the first, the need for storing earlier observations disappears. In that special case, if the elementary functions are all (nonhidden) model neurons, our method reduces to something nearly equivalent to the work of Almeida (1987). (Unfortunately, I have yet to obtain Pineda (1987), which may also be related.)

This section will begin with a review of our earlier approach, used when only the first kind of recurrence was present. Then we will propose a method for dealing with Grossberg/Hopfield recurrence. This method will

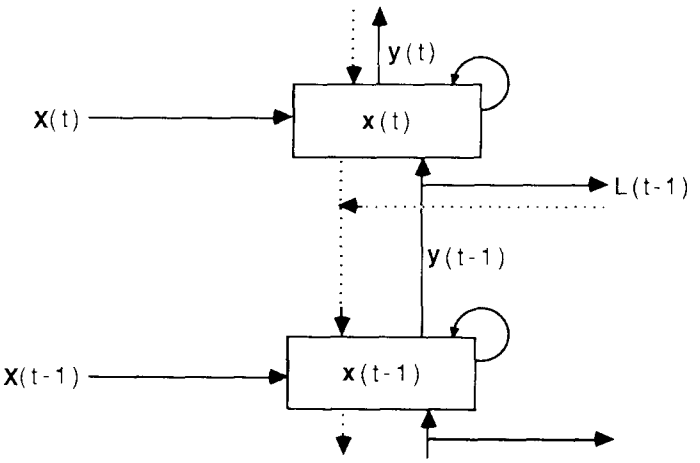

FIGURE 4. Proposed approach to recurrent networks.

be expressed in general form, allowing for both kinds of recurrence. There will actually be two variants of the method, one where the recurrence is "solved for" by an equation-solving system, and one where continuous-time differential equations are assumed to accomplish the same result.

The methods discussed here were derived as a generalization of dynamic feedback, though the aggregate form could have been derived as a generalization of the "adjoint" method used by Alsmiller et al. (1981), which I was aware of at the time. Section 4 will demonstrate an application of the aggregate form of this method.

Review of Classical, External Time Lags

This subsection will present a formulation of dynamic feedback which is technically a special case of (9). However, new notation will be introduced in order to make the time dependencies in Figure 4 more explicit.

Let us assume the existence of a network which implements a functional relationship **F**:

$$\mathbf{y}(t) = \mathbf{F}(\mathbf{y}(t-1), \mathbf{x}(t), t), \quad t = 1 \text{ to } T \tag{13}$$

where **y** has n components and **X** has m components. The function **F** is still assumed to depend on the parameters **b**, but there is no need to display that dependence explicitly here. We no longer need to put a caret over the output vector, **y**, because the true target vectors will be left implicit; in fact, the discussion here will assume an arbitrary differentiable function $L(\mathbf{y}(t),t)$, which may or may not have anything to do with forecast error or matching error. Our goal will be to calculate the derivatives of L', defined as the sum of L across all times t. (Actually, the calculations will work even if we only know the derivatives of the function, rather than the function itself.)

The vector **y** in (13) would typically include *both* the external outputs of the network *and* a set of auxiliary variables which serve as a kind of memory from one time period to the next. In some applications, such as economic forecasting, the auxiliary variables will sometimes be filtered representations of unknown, external variables. In applications like real-time control systems, where the external time lag may be less than a second, the auxiliary variables may represent a kind of reverberating short-term memory as described by Hebb (1949). In some applications, there is no need for auxiliary variables at all.

In order to represent **F** as a network, we will assume that:

$$x_j(t) = f_j(x_{1(t)}, \ldots, x_{j-1}(t)), \tag{14}$$

where the $x_i(t)$ are components of a vector $\mathbf{x}(t)$ which represents the total set of variables available as inputs or outputs to the network. Equation (14) looks like (7), superficially, but the references to a common time, t, make this a more specialized formulation; Equation 7 allowed for any variable $x_j(t)$ to receive inputs from any variables at earlier times.

In parallel with (7), we will again assume that the external inputs are in the front of this network:

$$x_i(t) = X_i(t), \quad i = 1 \text{ to } m \tag{15}$$

but we will leave the parameters **b** implicit for now. We will also assume that the components of $\mathbf{y}(t-1)$ come next in the **x** vector, followed by h hidden units, such that:

$$x_{m+i}(t) = y_i(t-1) \quad i = 1 \text{ to } n, \tag{16}$$

and j in (11) is assumed to run from $m + n + 1$ to $m + n + h + n$, so that:

$$y_i(t) = x_{m+n+h+i}(t) \tag{17}$$

Strictly speaking, if we assume that $h = 0$, we would arrive at a more general-looking structure, more like (7) and (9); however, this formulation gives us the freedom to set h greater than zero, which will be useful when dealing with continuous-time systems.

Next, in order to calculate all the derivatives of L', we have a choice of two approaches. Both approaches force us to think of $x_i(t)$ across all t and all i as the system of variables, with variables later in time always having a higher implied index (when we apply Equation 9) in this greater system. (See Werbos, 1974, 1988a for some simple examples.)

In the first approach—the aggregate approach—we treat (13) as the equation of the system, and treat the components of the vector **F** as the elementary functions. In this case, the application of (9) yields:

$$Z_k(t) = \frac{\partial L}{\partial y_k}(t) + \sum_{j=1}^{n} Z_j(t+1) F'_{jk}(t+1), \tag{18}$$

where $Z_k(t)$ will contain the ordered derivative of L' with respect to $y_k(t)$, where the equation is to be evaluated first for $t = T$ (for all k), then $T - 1$, and so on, where the rightmost (summation) term is to be treated as zero for $t + T$, and where F'_{jk} is the derivative of F_j (the jth component of **F**) with respect to y_k. The parameter derivatives—what we really need in most applications—also follow directly from (9):

$$\frac{\partial L'}{\partial b_i} = \sum_{t=1}^{T} \sum_{j=1}^{n} Z_j(t+1) \frac{\partial F_j(t+1)}{\partial b_i} \tag{19}$$

Superficially, (18) may appear as complex as (9) itself (since the sum over the right-hand side includes the ordered derivatives, $Z_j(t+1)$, for *all* subsequent variables in the system at *all* times); however, there are many applications (like Section 4) where each parameter b_i will appear in only one or a few of the functions F_j, so that the summation over j may be very sparse. (Note that we still assume that **F** depends on the parameters **b**, as discussed above, even though this de-

pendency has been implicit until now.) The summation over time can be carried out efficiently as a running sum, backwards in time, in parallel with (18); we will show how this can be done in (35c), of Section 4, *after* we have derived a version of this equation for use with recurrent systems. This method of calculating derivatives is very similar to earlier methods from control theory.

In the second approach (which is far more efficient for true networks), we treat (14) through (17) as the equations of the system, and we treat the f_j in these equations as the elementary functions. This approach (used in Werbos, 1974) is essentially the same as the RHW approach for recurrent networks, except that we feed the derivatives back to handle external time intervals instead of internal cycle times in iteration.

Applying (9) to the set of all variables over space and time yields the following set of equations to be evaluated as a set, first for all variables at $t = T$, then for $t = T - 1$, and so on:

$$z_{m+n+h+k}(t) = \frac{\partial L}{\partial y_k}(t) + \sum_{j=m+n+h+k+1}^{m+n+h+n} z_j(t) \frac{\partial f_j}{\partial x_{m+n+h+k}}(t) + z_{m+k}(t+1), \quad k = n \text{ to } 1 \qquad (20)$$

where the rightmost term is assumed zero for $t = T$, and

$$z_k(t) = \sum_{j=k+1}^{m+n+h+n} z_j(t) \frac{\partial f_j}{\partial x_k}(t) \quad k = m+n+h \text{ to } 1. \qquad (21)$$

Here the variables $z_k(t)$ refer to the ordered derivative of L' (the sum of L over time) with respect to $x_k(t)$. As with (9), these equations can be implemented efficiently on a parallel computer to the extent that the original system (Equation 11) could be. In deriving (20) from (9), I am accounting for the fact that $y_k(t)$ can have a direct causal effect (in our assumed network) only on *some* of the later variables—L itself (which yields the first term), $y_J(t)$ variables for J greater than k (generating the second term), and $x_{m+k}(t+1)$ (which by definition *equals* $y_k(t)$, so that the conventional partial derivative of $x_{m+k}(t+1)$ with respect to $y_k(t)$ is just 1). Likewise, (21) has only one term because these components of $\mathbf{x}(t)$ can only have a direct effect on later components of $\mathbf{x}(t)$.

Finally, the derivatives with respect to parameters or weights follow directly from (9), and may again be read off as running sums:

$$\frac{\partial L'}{\partial b_i} = \sum_{t=1}^{T} \sum_{j=t+1}^{m+n+h+n} z_j(t) \frac{\partial f_j}{\partial b_i}(t) \qquad (22)$$

The summations in (20) through (22) are generally very trivial and sparse, so long as the functions f_j only have a few inputs each.

These equations can be implemented most quickly if *all* components of $\mathbf{x}(t)$ have been stored for all t; however, the time cost is increased by only a fixed percentage (circa one-third?) if only the lower-order components are stored and the others regenerated at each time t while going backwards. (At most, x_1 through x_{m+n} are needed, but less will be needed if some components of $\mathbf{y}(t-1)$ are for external use only; to exploit this, it would help to locate such components in a block with higher index numbers in the vector $\mathbf{y}$.)

Networks Containing Both Types of Recurrence

Equation (13) may be generalized still further as:

$$\mathbf{F}(\mathbf{y}(t), \mathbf{y}(t-1), \mathbf{X}(t), t) = 0, \quad t = 1 \text{ to } T \qquad (23)$$

where $\mathbf{F}$ has the same dimensionality as $\mathbf{y}$. As before, the parameters $\mathbf{b}$ are still arguments of $\mathbf{F}$, but are left implicit. A system of this sort can yield forecasts only if we have some method available to solve these equations for $\mathbf{y}(t)$ when we are given values for $\mathbf{y}(t-1)$ and for $\mathbf{X}(t)$ and $\mathbf{b}$. In econometric forecasting, this system of equations (one equation for each component of $\mathbf{F}$) may be typed into a software package such as Troll (MIT, 1980) or SAS (1986), which then generates the forecasts. In neural nets, (23) may result from a hybrid continuous/discrete system such as:

$$\dot{y}_i(t) = F_i(\mathbf{y}(t), \mathbf{y}(t-1), \mathbf{X}(t), t), \qquad (24)$$

where we count on the mechanisms described by Hopfield and Tank (1986) to move $\mathbf{y}(t)$ quickly to a solution which fits (23). (Hopfield's mechanism works only for symmetric networks, but—after establishing definitions in Equations 26—I will cite an older, more general criterion, which is useful here but often difficult to apply.) In neurological terms, (24) would reflect the idea that *some* cells have a totally continuous-time response while others are partly controlled by some kind of clock pulse.

Before we can calculate the ordered derivatives of L', using either (18) or (9) directly, we need to calculate the *direct* causal impact of changing $\mathbf{y}(t-1)$ on $\mathbf{y}(t)$, when holding $\mathbf{b}$ constant; in other words, we need to know what corresponds to the matrix F'_{jk} in this situation. To calculate this, we may begin by taking the total differential of (23), which yields:

$$G(t)d\mathbf{y}(t) + H(t)d\mathbf{y}(t-1) = 0 \qquad (25)$$

where

$$G_{ij}(t) = \frac{\partial F_i(\mathbf{y}(t), \mathbf{y}(t-1), \mathbf{X}(t), t)}{\partial y_j(t)} \qquad (26a)$$

$$H_{ij}(t) = \frac{\partial F_i(\mathbf{y}(t), \mathbf{y}(t-1), \mathbf{X}(t), t)}{\partial y_j(t-1)}. \qquad (26b)$$

Likewise, to use (19), we will need to calculate the causal effect on $\mathbf{y}(t)$ of changing $\mathbf{b}$ while holding $\mathbf{y}(t-1)$ constant; this will be based on an equation like (25) but with $H(t)d\mathbf{y}(t-1)$ replaced by $J(t)d\mathbf{b}$, where we define:

$$J_{ij}(t) = \frac{\partial F_i(\mathbf{y}(t), \mathbf{y}(t-1), \mathbf{X}(t), t)}{\partial b_k}. \quad (26c)$$

Note that the matrix G will normally be nonsingular if (as we must assume) the original system in (23) can be solved for over a range of different values of $\mathbf{y}(t-1)$. In fact, for continuous-time systems, (24) can only converge if the eigenvalues of G all have negative (nonzero) real parts.

Equation (25) leads to:

$$d\mathbf{y}(t) = -G^{-1}(t)H(t)d\mathbf{y}(t-1). \quad (27)$$

In formal terms, this is just a linear dynamic system (albeit in infinitesimal quantities), which can be treated as a special case of (13). In fact, if we had taken the total differential of (13), we would have arrived at a linearized dynamic equation exactly like (27), except that the matrix F'_{jk} used in (18) would have replaced $-G^{-1}H$. Therefore, in this special case, F'_{jk} of (18) corresponds to the jk component of $-(G^{-1}H)$. By similar reasoning, the rightmost term of (19) corresponds with $-(G^{-1}J)$. With these two substitutions into (18) and (19), and minor changes to express the results as vector equations, we arrive at:

$$\mathbf{Z}(t) = \nabla_{\mathbf{y}}L(t) - H^T(t+1)(G^T(t+1))^{-1}\mathbf{Z}(t+1) \quad (28a)$$

$$\nabla_{\mathbf{b}}L' = \sum_t -J^T(t+1)(G^T(t+1))^{-1}\mathbf{Z}(t+1), \quad (28b)$$

where $\nabla_{\mathbf{y}}$ represents the vector of derivatives with respect to the components of $\mathbf{y}(t)$, where the recursion is from $t = T$ backwards again, and where the rightmost term in (28a) is treated as zero for $t = T$.

As a practical matter—either in neuron networks or econometric models—we do not have the inverse $(G^T(t))^{-1}$ available. Therefore, (28) cannot be used directly as a recursion rule to calculate the derivatives. However, we can overcome this problem simply by defining an auxiliary vector $\mathbf{w}(t)$:

$$\mathbf{w}(t) = -G^T(t)^{-1}\mathbf{Z}(t). \quad (29)$$

When we solve for $\mathbf{Z}(t)$ in this equation, and substitute the resulting expression for $\mathbf{Z}(t)$ into Equations (28) (while uniformly shifting back the time index in Equations 28), we arrive at the following equations which *can* be used to calculate the derivatives (if we invoke them in backwards time):

$$\mathbf{Z}(t) = -G^T(t)\mathbf{w}(t) \quad (30a)$$

$$\mathbf{Z}(t-1) = \nabla_{\mathbf{y}}L(t-1) + H^T(t)\mathbf{w}(t) \quad (30b)$$

$$\nabla_{\mathbf{b}}L' = \sum_t J^T(t)\mathbf{w}(t). \quad (30c)$$

Equation (30a) still must be solved for $\mathbf{w}$; in other words, $\mathbf{Z}(t)$ is the input to the required calculation and $\mathbf{w}(t)$ is the output. However, this can be done by the same mechanism used to solve (23) in the first place. This leads to two versions of the method, depending on the original solution method.

With econometric models, or other systems solved on a computer by an equation-solving package, the procedure is very straightforward. One can simply write down Equations (30) explicitly, and insert them into the same equation-solving package. Section 4 will provide an example of how one can do this, for a moderately large model. With fully recurrent nets, *unlike* (18) and (19), the aggregate formulation can be just as efficient as the network formulation, *if* we take care to break down the equations of the model into elementary relationships (whose sparsity will be accounted for in a good equation-solving package); in other words, we can expand the vector $\mathbf{y}$ to include the intermediate variables, and to enforce the sparsity of the matrices G, H, and J. In this manner, a single-layer recurrent network can represent anything that a multilayer feedforward network can. (Strictly speaking, however, we did break down a few of the longer model equations in our application, as will be discussed.) Notice that (30b) and (30c) are really just a conventional derivative calculation, as in conventional backpropagation (Equation 18), using $\mathbf{w}$ in place of $\mathbf{Z}(t+1)$.

With true continuous-time neural networks, based on (24), we need to formulate a network representation, translate (30b) and (30c) into a network version, and then use a continuous-time procedure to solve for the vector $\mathbf{w}$. To define the network itself, we may continue to use (14) through (17) with the proviso that those functions $f_k(t)$ which represent components of the vector $\mathbf{y}(t)$ may also include a dependency on any other component of the vector $\mathbf{y}(t)$, regardless of whose index is the greatest.

To translate Equations (30) into a network version, we begin by finding a continuous-time version of (30a) which, at any time t, is the first equation to be invoked when calculating derivatives. We cannot *deduce* a continuous-time version from (30a), but we can deduce that the following equation yields a solution for $\mathbf{w}$ which is equivalent to that of (30a):

$$\dot{w}_k(t) = z_{m+n}(t+1) + \sum_{j=1}^{n} \frac{\partial f_{m+n+h+j}}{\partial v_k}(t)w_j(t) \quad (31)$$

This equation reaches equilibrium when the time-derivative on the left is zero, which requires that the right-hand side be zero. However, the rightmost (summation) term in this equation is really just $G^T\mathbf{w}$, because the derivatives in that term correspond to our original definition of G applied to this case; therefore, the right-hand side of (31) will equal zero only when (30a) is satisfied. We already know that the eigenvalues of G must all have negative real parts, in order for (24) to converge in the first place; therefore, we may be sure that (31)—which is linear in the feedback variables—will also converge.

Next, we can translate (30b) into a network version quite easily if we exploit our understanding of causal flows. We may begin by replacing (23) by:

$$z_{m+n+h+k}(t) = w_k(t) + \frac{\partial L}{\partial y_k}(t), \quad k = 1 \text{ to } n. \tag{32}$$

Strictly speaking, this does not match (30b) exactly. The L term matches, but the rightmost term in (30b) would be zero in this case, and the $w_k(t)$ term requires explanation. The point is, if we go on to use (21) and (22) as they stand, after having invoked (32), we will get the correct ordered derivatives for the inputs, because (30b) and (30c) both require that we add in feedback from $\mathbf{w}$. In fact, (30b) and (30c) do not show feedback from the L-derivative through H^T to the lower-order derivatives; however, this was due to the exclusion of indirect impacts at time t in the aggregate version of the net; Equations (20) through (22) should make it clear that we do want to account for such effects when the network specification permits them.

In summary, our continuous-time procedure would go backwards in time, t, to calculate the derivatives. At each time t, it would first invoke (31) through to equilibrium, and then invoke (32), (21), and (22), in that order, going backwards from later variables to earlier variables. A hybrid approach would continue to use an equation-solver to solve (30), and then proceed to (32), (21), and (22).

Strictly speaking, we can generalize this arrangement still further by allowing hidden units as well to depend on components of the vector $\mathbf{y}(t)$. In order to implement (30a), we then need to allocate another vector $\mathbf{z}'$ of length h, to establish the convention that $z'_{h+k}(t)$ refers to $w_k(t)$, to add the equation:

$$z'_k(t) = \sum_{j=1}^{h+n} z'_j(t) \frac{\partial f_{m+n+j}}{\partial x_{m+n+k}}(t), \tag{33}$$

and to add the following term to the right-hand side of (31):

$$\sum_j \frac{\partial f_{m+n+j}}{\partial y_k(t)} z'_j(t). \tag{34}$$

This generalization will not be considered further, because its value is questionable in neural applications; however, it is necessary to use this generalization, in principle, to describe what we did in our application in Section 4 (when "hidden variables" were defined in order to break up a few big equations).

In both versions of the method—equation-based and continuous—the difficulty of solving for $\mathbf{w}$ depends on the eigenvalues and sparsity of the matrix G; since this is also true for the forwards version of the system (Equations 23 or 24), but the forwards version is nonlinear, the calculation of $\mathbf{w}(t)$ should never be more expensive than the calculation of $\mathbf{y}(t)$ in the forwards system. Likewise, the cost of running (31), (21), and (23) should be comparable to the usual costs of running conventional backpropagation through one iteration. The storage costs (aside from the need to store one additional vector, $\mathbf{w}$) are the same as those of backpropagation in a network without the continuous-time recurrence. In the case where $\mathbf{y}(t-1)$ is not actually used in (23) or (24), so that continuous-time recurrence is the only form of recurrence, there is no need to store any information at all from earlier times t.

Issues Related to Real-Time Adaptation

Two different adaptation strategies are now used with backpropagation, when adapting artificial neural networks. Both strategies involve iterating through the data base or training set many times, until the estimated values of the weights settle down or the level of error is acceptably small.

Hinton (1987) calls one of these strategies "batch learning." In batch learning, each iteration begins with a calculation of the derivatives of error with respect to the weights, summed up over all patterns exactly as indicated in (19). The weights are then adapted in proportion to these derivatives (or by use of more sophisticated methods using the derivatives). Then a new iteration begins. Statisticians almost always use batch learning, as I have myself when using three-net type architectures. The best convergence rates I have seen so far with artificial neural networks have involved the use of batch learning and sophisticated numerical methods, even when an $O(n)$ storage constraint is imposed and the higher cost per iteration is accounted for.

The other strategy I usually call pattern learning. In pattern learning, one does not wait to calculate the entire sum in (19) before adapting the weights. One calculates the component of (19) for pattern number t, adapts the weights immediately, and then moves on to the next pattern. This kind of approach can be used with continuous-time or simultaneous-time recurrent networks, exactly as it can with feedforward networks. When external time-lags are present, however, pattern learning leads to an inconsistency between the values of $\mathbf{y}(t)$ currently available and those implied across all time by the new set of weights, after the weights are adapted for a given observation; as a result, big learning rates could lead to a failure to converge in some cases. However, there are similar problems which can lead to divergence even when pattern learning is used to adapt feedforward networks. (With external time-lags, the problem might be reduced by adapting weights only during the backwards pass, i.e., backwards through the set of patterns.) Because of the current need for small learning rates, convergence times have been very long with pattern learning, even with feedforward networks; however, this merely underlines the need for further research, to adapt the methods of numerical analysis and to combine the power of backpropagation and content-addressable memory (Werbos, 1988b).

Natural systems, like the human brain, do not use

batch learning *or* pattern learning. Instead, they use real-time adaptation, in which each pattern is available only once, and then lost (except for its impact on the weights and on short-term memory). The patterns are experienced in forwards time only. For feedforward networks, this is really the same as pattern learning, except that only one pass through the database is allowed. The same situation applies to networks with simultaneous/continuous recurrence only. However, when external time-lags are present, our recurrence formulas simply cannot be applied exactly in real-time adaptation, because of the lack of a database to go back through. A similar problem would apply to continuous-time systems which implement a similar recurrent, short-term memory and which therefore violate the conditions on G given above. (This violation follows from the fact that systems which obey our conditions allow one to solve for the equilibrium system state as a function of present inputs only.) In either case, true real-time adaptation would require the use of some sort of approximation.

The easiest and least accurate approximations would simply cut off feedback to earlier than one or two observations into the past. The accuracy of such approximations may depend on the loss functions actually used, in a complex way. Far better, in theory, is to treat the determination of $y_i(t)$ as a long-term optimization problem, as if $\mathbf{y}(t)$ were a vector of actions (like $\mathbf{u}(t)$ in Figure 2) chosen to as to minimize the sum of prediction errors over present and future time. To apply the optimization methods mentioned in Section 2, note that prediction error is normally represented as a sum of distinct components (i.e., errors on individual variables). Also note that there is no need for an additional predictive model; the equations of the existing network specify exactly how $\mathbf{y}(t)$ affects $\mathbf{y}(t+1)$, and so on. Implementing the optimization methods of Section 2, we would create something like an estimate J_i which would serve as a direct, local source of feedback for each component y_i of the $\mathbf{y}$ vector. The details of this possibility are beyond the scope of the present paper; however, since the action variables, the dynamics of the system, and the utility measures have all been specified, it should be straightforward in principle to work out these details. Furthermore, since these optimization methods all impose costs on the order of $O(N)$—like backpropagation itself—this should be a workable approach.

This approach should not be confused, again, with the use of Figure 2 to optimize overt actions. This approach could be used with any recurrent net, emerging from the architecture of Figure 1 or from other architectures. When this approach is used to help adapt the nets shown in Figures 1 and 2, then the J network used to give feedback to the $\mathbf{y}$ variables would be quite distinct from the J network used to adapt overt actions.

4. DESCRIPTION OF THE APPLICATION

Background and Goals

The work reported here was performed in 1982 for the Energy Information Administration (EIA), prior to the construction of a new natural gas supply model. It has never been published, since the results were mainly for internal use. To our knowledge, this was the first successful, operational test of Equations 30 in calculating the derivatives of a fully recurrent system.

The purpose of this project was to better understand the properties of EIA's previous model of natural gas markets, the Natural Gas Market Model (NGMM), which had been used in a major study of natural gas deregulation (McNicol, O'Neill, & Dickens, 1981). The first stage of this project was simply to penetrate the code of the model, and convert 1500 lines of FORTRAN into an explicit, equivalent 73-equation system in Troll, corresponding exactly to a 73-component vector $\mathbf{F}$ in (7). A concise, consolidated description of the model was then published (Werbos, 1981). The model was then updated to an 83-equation system to reflect more recent information on natural gas availability by regulatory category (O'Neill & Dickens, 1981) and more recent demand forecasts (EIA, 1982).

The major goal of this project was to evaluate what really drove the forecasts of the model. The model was a highly interactive system, dependent on dozens of uncertain parameters and initial values. To vary all of these parameters and all of the variables of the model, in all years, would have required hundreds of runs of the model. It was easier and more accurate to create an "adjoint model"—replicating the feedback calculations implied by Equations (30)—which would yield the derivatives of a selected model result L with respect to *all* parameters and all variables in all years in only one run. In other words, dynamic feedback was used here simply to calculate derivatives, which were of interest in their own right as a diagnostic tool in evaluating the model. In principle, this kind of sensitivity analysis could also be used to locate policy levers which are especially important in changing future outcomes.

Implementation of Dynamic Feedback

The analysis here was carried out in Troll (MIT, 1980), a standard software package developed by the MIT Center for Computational Economics and Management Science. An "adjoint model" was created in Troll, representing exactly the calculations implied by Equations 30.

Troll, like most dynamic modeling packages, only allows calculations forwards in time. Therefore, Equations (30) had to be translated into an equivalent set of equations running in reverse time. We defined $t' = 1990 - t$, and re-expressed Equations (30) in terms

of t'. For convenience, we assumed that $L(t) = 0$ for t less than the terminal year, 1990. (However, a running total was created to handle the one instance where we were interested in what influenced the sum of a variable over time, as opposed to its 1990 value.) Also, because we had not yet considered what conventions would lead to felicitous notation with continuous-time neural nets, we defined $\mathbf{w}(t)$ as minus $\mathbf{w}(t+1)$ (where the latter copy of $\mathbf{w}$ is defined as in Equation 29). Substituting these definitions for t' and $\mathbf{w}$ into Equations (30), we arrive at the equations actually implemented in the application:

$$\mathbf{Z}'(t'-1) = G^T(t'-1)\mathbf{w}(t') \tag{35a}$$

$$-\mathbf{Z}'(t') = H^T(t'-1)\mathbf{w}(t') \tag{35b}$$

$$\mathbf{a}(t') = \mathbf{a}(t'-1) - J^T(t'-1)\mathbf{w}(t'), \tag{35c}$$

where $\mathbf{a}(0)$ will contain the final vector of derivatives of L with respect to the parameters.

In order to implement Equations (35), we followed a straightforward procedure that could be implemented quite easily in a package such as Troll. (This was verified in 1981 when it was proposed to the developers of Troll, in connection with an ongoing contract with the Department of Energy; unfortunately, other priorities preempted this option.) To understand this procedure, it would help to consider an example, based on a simplified version of a few of the model's equations:

#25: exploration(t)
$= b_1 * (\text{exploration}(t-1))^{b2} * (\text{gas\_\_price}(t)/\text{drill\_\_price}(t))^{b3}$

#26: cumulative__exploration(t)
$= \text{cumulative\_\_exploration}(t-1) + \text{exploration}(t)$

#49: drill__price(t) + drill__price$(t-1)$
$+ b_4 + b_5 * (\text{rig\_\_use}(t)/(1 - \text{rig\_\_use}(t))$

#67: industry__demand(t)
$= \text{base\_\_demand}(t) * (\text{gas\_\_price}(t)/\text{base\_\_price}(t))^{b6}$
$* (\text{industry\_\_demand}(t-1)/\text{base\_\_demand}(t))^{b7}$

The Troll equation numbers (between 1 and 83) are shown on the left. The first three equations describe how the utilization of drill rigs affects changes in the price of drilling, which in turn combines with the price of gas to affect exploration for gas. The last equation shows how interstate industrial gas demand will differ from a previous baseline forecast, if the actual gas price differs from the (base) price assumed in that forecast. The model solves to find a price which matches supply and demand. Notice how values of $b2$ and $b7$ near zero would make the forecasts dependent on conditions in the present time, while values near one would tend to yield a kind of exponential growth process (because outside factors then determine the *rate of growth* of the variables being projected, instead of their actual values.)

Our first step, in creating an adjoint model, was simply to write out all the component equations implied by (35a). To do this in a comprehensible way, we adopted a naming convention in which, for example, Fexploration(t) corresponded to $Z_{25}(t)$. However, we simply used $W25(t)$ to represent $W_{25}(t)$. Following this convention, we can calculate Fexploration$(t'-1)$ as implied by (35a) by looking through *all* the equations and looking for occurrence of the variable "exploration(t)"; if we find one in equation j, we calculate $G^T_{25,j}$ by simply differentiating the equation with respect to exploration(t). If exploration(t) appears on the left-hand side of an equation, we treat that as an appearance on the right-hand side with a minus sign. Applying this procedure to the example above, we get:

Fexploration$(t'-1)$
$= W25(t) * (-1) + W26(t) * (+1) + \cdots$

Fdrill__price$(t'-1)$
$= W25(t') * (\text{exploration}(t'-1) * (-b_3/\text{drill\_\_price}(t'-1)) + W49 * (-1) + \cdots.$

In the first of these equations, the (-1) simply came from differentiating Equation #25 *after* exploration(t) is moved to the right-hand side. The $(+1)$ came from differentiating the right-hand side of Equation #26 with respect to exploration(t). The next equation came from a similar calculation; however, note that all references to variables convert t to $t'-1$ and $t-1$ to t', because of the time reversal. The triple dots here refer to other terms which involve the differentiation of other equations, not given in our example. Mechanically, it was easier to do all this by writing "Fname$(t'-1) =$" for each variable, on a separate line of a large sheet of paper, and going through the list of equations in order, looking for all unlagged variables and adding terms to their equations.

Equation (35b) was handled essentially the same way, except that we looked for lagged references (i.e., to variable $(t-1)$), differentiated with respect to lagged variables, and began the relevant equations with (for example) "Fexploration$(t') =$." Equation (35c) was likewise straightforward. After completing this exercise, we simply typed the set of equations into Troll, and asked Troll to solve the set of equations from $t' = 1$ through $t' = T$. (This also required the use of a few Troll instructions to create a database made up of the original model variables, reversed in time.)

All of these tasks were completed in about two days. However, because the approach was new, two weeks were then used mainly to test, but also to debug the results. Modified versions of the model and of its adjoint were created in which the free market price of gas was made exogenous, so that the flow of causation and calculated feedback could be compared at all points in the model. Checks against derivatives by brute force

TABLE 1
Parameters with the Five Biggest Impacts on 1990 Average Residential Gas Price (i.e., with L' = Gas Price)

Description of Parameter	Total Impact	Parameter Value
Elasticity of exploratory gas drilling to its previous value (i.e., *b2* in equation labelled "#25" above)	$150	.8
Elasticity of gas development drilling to its previous value	$25.1	.8
Elasticity of oil development drilling to its previous value	$18.8	.9
Impact of oil production ($t - 1$) on oil production (t)	$15.6	.9
Impact of nonassociated gas production ($t - 1$) on itself (t)	$14.6	.9

parameter shifts and variable shifts were used; these required trying several step sizes (at least plus and minus some amount), because of problems with rounding error and nonlinear effects with the brute force method. At this point, the adjoint method has passed very severe tests of its accuracy. The adjoint, unlike brute force methods, is also "well-conditioned" numerically; the reason for this, technically, is that the transpose of $G^{-1}H$ has the same "condition number" as $G^{-1}H$ itself (Forsythe & Moler, 1967), so that the adjoint is as well conditioned as the original model itself.

Results of the Analysis

Tables 1 through 3 below summarize the results of greatest interest.

Table 1 provides a rank-ordering of the five most important "items" input to the model, where "items" include both parameters and initial values in principle. Importance is measured in terms of "Total Impact," defined as the change in residential gas prices which would result from setting the item to zero (assuming no change in the derivative). From an economist's point of view, the "Total Impact" as defined here is just the elasticity of gas prices with respect to each item, multiplied by the base case residential gas price for 1990. Out of the 35 most important items only two involved the demand for natural gas, and two involved initial values for 1979; thus the results of the model were clearly driven by supply-side assumptions.

Six other 1990 outcome variables were also examined with the same adjoint model: (a) DEMAND, total U.S. wellhead gas demand in quadrillion Btu; (b) CUMEXTRA, cumulative supplemental gas (potential shortages) over 1979–1990; (c) PSUPPLY, the unregulated wellhead price of gas; (d) RN.NAGAS, proved reserve balance of nonassociated gas; (e) SUSGASB, free-market domestic gas production; and (f) SDEEP, U.S. production of gas from 15,000 feet or deeper. The results with these other measures of outcome were similar to those of Table 1, but even more tilted towards the supply-oriented items input to the model.

The adjoint model also printed out information about the dynamics of the effect of each item, as shown in Table 2. The 1979 row of Table 3, like all the numbers in Table 1, reports the impact of changing the item on changing the outcome variable. This derivative essentially answers the usual question: "If you change this

TABLE 2
Ordered Derivatives of DEMAND with Respect to Three Items Over Time

	Elasticity of Gas Exploration to its Past (*b2* in Equation #25)	Ratio for Gas to Oil in Oil Production	Industrial Demand Lag Factor (*b7* in Equation #67)
Parameter Value	.806	.735	.69
Ordered Derivatives ($z_i(t)$) From Year . . .			
1990	0	0	0
1989	−.5	2.5	−.3
1988	1.8	2.1	−.4
1987	9.0	1.7	−.6
1986	22.0	1.3	−.7
1985	40.0	1.0	−.8
1984	63.0	0.7	−.9
1983	89.0	0.6	−.9
1982	118.0	0.5	−.9
1981	150.0	0.4	−.9
1980	184.0	0.2	−.9
1979	222.0	−.9	−.9

input to the model by one unit, while keeping all the other inputs as they are, how much will the outcome change?" The 1985 row, however, answers the question: "If you changed this parameter by one unit in 1985, and afterwards, but used the old value for it before 1985 (still holding the other items constant), how much would the outcome change?" (Likewise, for variables, the 1985 row reports the impact of an "autonomous change" in 1985, such that the other variables in 1985 are unchanged.) The main purpose of this table is to illustrate the diagnostic value of ordered derivatives in understanding how any system behaves over time.

Table 2 shows clearly that the gas exploration elasticity acquires its importance because of its cumulative effect over time. Like the population growth rate in population forecasting, this parameter has a greater impact on the forecast as the forecast interval grows. For the same reason, random errors in estimating this parameter will lead to cumulative errors in forecasting almost any of the outputs of the model.

The impact of the industrial demand lag term also grows with time. This parameter, like the exploration elasticity, is an "inertia" term; it indicates how much industrial gas demand is affected by its own past value. However, after a couple of years, the rate of growth of its importance is greatly damped. This would appear to mean that the "memory" on the demand side is quickly overshadowed by the impact of current prices on demand, so that the accumulation of impact is reduced. This pattern of a declining rate of growth in impact applies to most parameters in the model; impact grows with time, though at a decelerating rate, because more time simply adds to the points where the parameter affects the system, usually pushing the system in the same direction. The rate of growth declines for most but not all parameters, because of systems dynamic effects beyond the scope of this paper; if the system under study were "stationary," however (always decaying to the same equilibrium no matter what its initial state), then the rate of growth would eventually decline for all parameters.

Finally, Table 2 shows that the "ratio for gas to oil in oil production" declines steadily in importance as we go back to 1980. Steady decline is typical for the impact of a variable, but it is very unusual for a parameter. In this case, the direction of the immediate impact reverses between 1990 and the preceding years, perhaps because the direct impact in 1990 depends on the size of this parameter RELATIVE TO variables increased by the parameter before 1990. This parameter is odd in another way: in 1979, there is a sudden "blip" in the impact of this item, and of a few other items. This "blip" appears due to the equation which enforces a fixed price of drilling in 1980, when all the rest of the variables are treated as model predictions; changes in a parameter can force changes in the 1980 values of the variables which have nothing to do with the causal impact of the parameter, because of the equation forcing the 1980 drilling costs to not be changed accordingly. Note that this kind of startup problem affects the sensitivity of a model, regardless of what method is used to test the sensitivity; it can be a serious problem in interpreting the results of "what if" analysis based on changes in model assumptions, for many models.

TABLE 3
Results of Different Values for Elasticity Parameter (*b7*)

Elasticity (Exponent)	1990 Average Residential Gas Price in 1979 Dollars
.806	$7.30 (actual forecast)
.807	$7.13
.816	$6.18
.796	$8.23

All of these "sensitivity coefficients" theoretically represent the effect of small changes from the base case (continuation of the Natural Gas Policy Act). To verify the large-scale importance of the most important parameter—the elasticity of exploratory gas drilling to its own past value—the original model was rerun for four different values of the elasticity. These results are shown in Table 3.

According to the original model documentation, the standard error of this exponent was .08, much larger than the changes made here. This result did not necessarily invalidate the model, for reasons beyond the scope of this paper; however, it did lead to the conclusion that attention should be redirected towards the use of new statistical methods to estimate this kind of parameter more accurately, and towards possible respecification of the equations they appear in. The new methods cited in Section 2 *exploit* the phenomenon of cumulative error, and use multiyear tests of forecast error, in order to arrive at more accurate parameter estimates which lead to less cumulative error in forecasting. Previous research on these methods (Werbos, 1974, 1983a; Werbos & Titus, 1978) suggests that this model was far from unique in this weakness, but was typical of a large class of econometric models.

Conclusion

A generalization of dynamic feedback (the central component of "backpropagation") to deal with recursive ("simultaneous") time-dependent networks has been developed and tested, and has led to applications of importance to practical econometric forecasting. These applications in turn point to the importance of using new loss functions instead of regression when estimating many models; the use of these loss functions tends to require dynamic feedback for efficient, reliable implementation.

REFERENCES

Almeida, L. B. (1987). A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. *Proceedings of the IEEE First International Conference on Neural Networks, Vol. II* (pp. 609–618). New York: IEEE.

Alsmiller, R. G. et al. (1981). *Adjoint sensitivity analysis and its application to LEAP Model 22c.* ORNL/TM-7789. Oak Ridge, TN: Oak Ridge National Laboratory.

Athans, M., & Kalb, P. L. (1966). *Optimal control theory.* New York: McGraw-Hill.

Barto, A., Sutton, R., & Anderson, C. (1983). Neuron-like adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics,* **SMC-13,** 834–846.

Brode, J., Werbos, P., & Dunn, E. (1975). *TSP in the Datatran language.* Cambridge, MA: MIT Cambridge Project. (Also distributed by the MIT Information Processing Services Department and DOD/AFDSC as a contract product.)

Dennis, J., & Schnabel, R. (1983). *Numerical methods for unconstrained optimization and nonlinear equations.* Englewood Cliffs, NJ: Prentice-Hall.

Energy Information Administration. (1982). *1981 Annual report to Congress, Volume 3.* DOE/EIA-0173(81)/3. Washington, DC: National Energy Information Center (NEIC, 202-586-8800). Washington, DC: U.S. Department of Energy.

Foote, S., & Morrison, J. (1987). Extrathalamic modulation of cortical function. *Annual Review of Neuroscience,* **10,** 67–95.

Forsythe, G., & Moler, C. (1967). *Computer solution of linear algebraic systems.* Englewood Cliffs, NJ: Prentice-Hall.

Grossberg, S. (1976). Adaptive pattern classification and universal recording, II: Feedback, expectation, olfaction and illusions. *Biological Cybernetics,* **23,** 187–202.

Grossberg, S., Levine, D., & Schmajuk, N. (1987). Predictive regulation of associative learning in neural networks by reinforcement and attentive feedback. *Proceedings of the 1987 IEEE International Conference on Systems, Man and Cybernetics, Volume III.* (IEEE Catalog No. 87CH2503-3). New York: IEEE.

Hebb, D. O. (1949). *The organization of behavior.* New York: Wiley.

Hinton, G. (1987, June). *Connectionist learning procedures* (Tech. Rep. No. CMU-CS-87-115). Pittsburgh: Carnegie-Mellon University, Computer Science Department.

Hopfield, J., & Tank, D. (1986). Computing with neural circuits: A model. *Science,* **233,** 625–633.

MIT Center for Computational Research In Economics and Management Science. (1980). *Troll Users' Guide.* Cambridge, MA. IPS Technical Services, MIT Building 39-327.

McNicol, D., O'Neill, R., & Dickens, P. (1981). *Analysis of the economic effects of accelerated deregulation of natural gas prices.* DOE/EIA-0303. Washington DC: NEIC (see EIA 1980).

O'Neill, R., & Dickens, P. (1981). *An analysis of the natural gas policy act and several alternatives: Part 1.* DOE/EIA-0313. Washington DC: NEIC (see EIA 1980).

Parker, D. B. (1985). *Learning-logic* (Report TR-47). Cambridge, MA: MIT Center for Research in Computational Economics and Management Science.

Parker, D. B. (1987). Second-order backpropagation: Implementing an optimal $O(n)$ approximation to Newton's method in an artificial neural network. Unpublished manuscript.

Pineda, F. J. (1987). Generalization of backpropagation to recurrent and higher order networks. *Proceedings of the IEEE Conference on Neural Information Processing Systems.* New York: IEEE.

Rumelhart, D., Hinton, G., & Williams, R. (1986). *Parallel distributed processing* (Chap. 8). Cambridge, MA: MIT Press.

SAS Institute. (1986). *SAS users guide: Statistics.* Cary, NC: SAS Institute Inc.

Schmitt, F. O. (Ed.). (1970–1971). *Neurosciences* (1st and 2nd study programs). New York: Rockefeller University Press.

Werbos, P. (1968). Elements of intelligence. *Cybernetica,* 3.

Werbos, P. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* Ph.D. thesis Harvard University. (Also printed as a report of the Harvard/MIT Cambridge Project, 1975, under Dept. of Defense contract.)

Werbos, P. (1977). Advanced forecasting methods for global crisis warning and models of intelligence. *General Systems Yearbook,* **22,** 25–38.

Werbos, P., & Titus, J. (1978). An empirical test of new forecasting methods derived from a theory of intelligence: the prediction of conflict In Latin America. *IEEE Transactions on Systems, Man and Cybernetics,* **SMC-8,** 657–666.

Werbos, P. (1979). Changes in global policy analysis procedures suggested by new methods of optimization. *Policy Analysis and Information Systems,* **3,** 27–51.

Werbos, P. (1981). *The Natural Gas Market Model: Equations and data sources.* DOE/EIA-0355. Washington DC: NEIC (see EIA 1980).

Werbos, P. (1982). Applications of advances in nonlinear sensitivity analysis. In R. Drenick & F. Kozin (Eds.), *Systems modeling and optimization: Proceedings of the 10th IFIP Conference, New York* (pp. 762–777). New York: Springer-Verlag.

Werbos, P. (1983a). *A statistical analysis of what drives industrial energy demand* (Chap. 4, Section on Dynamic Robust Estimation). DOE/EIA-0420/3. Washington DC: NEIC (see EIA 1980).

Werbos, P. (1983b). Solving and optimizing complex systems: Lessons from the EIA long-term model. In B. Lev. (Ed.), *Energy models and studies.* New York: North Holland.

Werbos, P. (1986). Generalized information requirements of intelligent decision-making systems. In *SUGI 11 Proceedings.* SAS Institute: Cary, NC (A revised version, available from the author, is easier to read and contains more discussions of psychology.)

Werbos, P. (1987a). Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Transactions on Systems, Man and Cybernetics,* **SMC-17,** 7–20.

Werbos, P. (1987b). Learning how the world works: Specifications for predictive networks in robots and brains. In *Proceedings of the 1987 IEEE International Conference on Systems, Man and Cybernetics, Volume I* (pp. 302–310). (IEEE Catalog No. 87CH2503-1). New York: IEEE.

Werbos, P. (1987c). Backpropagation versus content-addressable memory: Applications, evaluation, and synthesis. Unpublished manuscript.

Werbos, P. (in press). Econometric techniques: Theory versus practice. In J. Weyant & T. Kuczmowski (Eds.), *Planning in a risky environment: A handbook of energy/economy modeling.* New York: Pergamon.

Werbos, P. (1988a). Maximizing long-term gas industry profits in two minutes in Lotus using neural network methods. *IEEE Transactions on Systems, Man and Cybernetics.*

Werbos, P. (1988b). Backpropagation: Past and future. *Proceedings of the IEEE International Conference on Neural Networks, 1988, Vol. I* (pp. 343–353). (IEEE Catalog No. 88CH2632-8). New York: IEEE.