BACKPROPAGATION: PAST AND FUTURE

by Paul J. Werbos, 8411 48th Ave., College Park, Maryland 20740*

ABSTRACT

From a quick reading of a few examples from Rumelhart, Hinton and Williams(RHW), some scientists have concluded that backpropagation is a specialized method for pattern classification, of little relevance to broader problems, to parallel computing, or to our understanding of the human brain. In a series of papers in process, I have questioned these beliefs, and proposed the development of a general theory of intelligence in which backpropagation (in my 1974 formulation [1,2]) and comparisons to the brain play a central role. I have also pointed to a series of intermediate steps and applications leading up to the construction of such generalized systems, including past applications to social science which in some ways go beyond the work in AI as such. This paper will try to give a condensed mathematical summary of that work (and hope that the papers cited and the talk clarify a few of the inevitable loose ends). This paper begins by summarizing a generalized formulation of backpropagation, and then discusses network architectures and applications which it opens up.

A GENERALIZED FORMULATION OF BACKPROPAGATION

Thanks to David Parker (whose work on backpropagation also preceded RHW), some of you may know that in 1972 I presented my Harvard thesis committee with a neuron-based form of backpropagation nearly identical to RHW's. Committee members ruled that this algorithm would be interesting as a seminar paper but not important enough to warrant a Ph.D. The idea was indeed simple and untested at that time, but simple ideas can be important. This paper will present other simple ideas which may be even more important than backpropagation.

For my thesis, I reformulated backpropagation as a general method for any feedforwards network of functions. The mathematics and neural applications were later discussed in some detail at a conference similar to this one in size and visibility[2], and cited elsewhere.

In my notation, an RHW network implements two equations:

$$net_i(t) = \sum_{j<i}^{N} W_{ij}\, s(net_j(t)), \qquad i=m+1,N+n \tag{1}$$

$$Error = \sum_{t=1}^{T} \sum_{j=1}^{n} \tfrac{1}{2}(y_j(t) - s(net_{N+j}(t))^2 \tag{2}$$

where t is the pattern number (out of T patterns), s is the sigmoid function, and where the first m components of "net" are inputs and the last n components represent predictions, in effect, of the target values $y_j(t)$. By fixing some W_{ij} to zero, one easily builds sparse networks which are efficient to use on parallel computers. The goal of backpropagation is to adapt the weights W_{ij} so as to minimize Error, for given values of the inputs and targets across all patterns t. Equation 1 is a special case of:

* Deep thanks to David Parker and Rich Sutton, whose help and encouragement made this possible. Thanks also to DOE for letting me write this in anticipation of a detail as NSF Program Manager for Neuroengineering when this paper is presented. This paper does not necessarily reflect official views of either agency.

$$x_i(t) = f_i(x_1(t),\ldots,x_{i-1}(t),\underline{x}(1),\ldots,\underline{x}(t-1),\underline{w},t), \qquad i=m+1,N+n \qquad (3)$$

where $\underline{w}$ is a vector of weights or parameters, f_i is any differentiable function, and $\underline{x}(t)$ is the vector formed by $x_1(t)$ through $x_{N+n}(t)$. Equations 1 and 2 (or 3 and 2) are special cases of:

$$x_i = f_i(x_j,\ j<i), \qquad i = M+1,\ N' \qquad (4)$$

where x_i is any quantity calculated in the entire system across space and time, and where M represents the total number of inputs to the system -- not only input variables at all times, but weights as well. For example, in an RHW net with m=4, N=10, n=2, T=30 and 25 weights, the number of total inputs would be M=mT+n weights=145. Thus the first quantity which is calculated, $net_5(1)$, would be represented in equation 4 as x_{146}; $net_6(1)$ would be x_{147}; $net_{12}(1)$ - the last quantity calculated for the first pattern - would be x_{153}; $net_5(2)$ would be next, as x_{154}, and so on. "Error" would correspond to $x_{N'}$, the last quantity calculated. Equation 4 would be cumbersome to use if we had to keep track of all these subscripts, but in practice we only need to keep track of which quantity gets calculated before which other quantity. Equation 4 is the most general possible form for a feedforwards system of differentiable functions over a finite number of quantities.

Critics have argued that backpropagation is nothing more than the use of the chain rule to calculate the derivatives of Error with respect to the weights in equations 1 and 2, followed by the use of steepest descent or other methods to adjust the weights. This is true, in a sense, but the ordinary chain rule does not do the job in a straightforward way. We have proven[1] a chain rule for "ordered derivatives" which may be summarized for now as the recurrence rule:

$$F\_x_i = \frac{\partial f_{N'}}{\partial x_i} + \sum_{j>i}^{N'-1} F\_x_j * \frac{\partial f_j}{\partial x_i}, \qquad i = N'-1,\ldots,1 \qquad (5)$$

where the partial derivatives refer to the derivatives of the functions f_i, expressed as simple functions of their direct arguments, while F_x_i is the total (ordered) derivative of the target variable (Error, in this case) with respect to x_i. To adapt the weights in the RHW system, we need to know F_W_{ij} for all the weights. Equation 5 cannot be inserted directly into a simple computer package, because it includes partial derivatives; however, we can generate equations to plug in simply by working out the partial derivatives and substituting them into equation 5. More precisely, we consider each quantity x_i calculated in the system (starting from the quantity last calculated), and look for that quantity on the right-hand-side of the system equations. For each equation it appears in, we differentiate the equation with respect to x_i, and plug that derivative into equation 5 (bearing in mind that "x_j" is the left-hand-side variable of the equation being differentiated). (See [3,4] for a longer tutorial.) For example, in equations 1 and 2, $net_i(t)$, for i greater than N, appears only in the right-hand side of equation 2 (because of the "N" over the summation sign in equation 1.). Differentiating Error with respect to net_i in equation 2, and plugging into equation 5, we get:

$$F\_net_i(t) = (s(net_i(t)) - y_{i-N}(t))\ s'(net_i(t)) \qquad i>N \qquad (6)$$

Likewise, $net_j(t)$, for j between m+1 and N, appears only on the right of equation 1. Differentiating $net_j(t)$ as expressed in equation 1 with respect to $net_j(t)$, we get a partial derivative of $W_{ij}s'(net_j(t))$; plugging into equation 5, we get:

$$F\_net_j(t) = \sum_{i>j}^{N+n} F\_net_i(t) * W_{ij} s'(net_j(t)) \qquad m < j \leq N \qquad (7)$$

Likewise:

$$F\_W_{ij} = \sum_t F\_net_i(t)\, s(net_j(t)) \qquad (8)$$

The backwards equations, 6 through 8, are essentially identical to the RHW formulas, when steepest descent is applied to the derivatives F_W_{ij}. The adaptation can be done on a batch basis (adding up the whole sum in F_W_{ij} in equation 8 before modifying the weights), on a pattern basis (adapting after evaluating each pattern t, and cycling through all the patterns) or on a real-time basis (cycling through each pattern only once, in real time). If some weights were fixed at zero, to make the net sparse or efficient to implement on a parallel computer, then equation 7 will inherit the same sparsity or suitability. An analog implementation would merely require a modulation/timing system, to enforce alternation between forwards sweeps and backwards sweeps. (In fact, the "second order" axo-axonal synapses of the cerebral cortex seem to involve this kind of modulation, rather than higher-order logic units[5].)

CAUSAL MODELLING/FORECASTING AND UNSUPERVISED FEATURE GENERATION

Equations 1 and 2 represent a standard supervised learning situation. If we define the input variable $x_j(t)$ as $s(net_j(t))$, for j between 1 and m, then these two equations really give us a recipe or model to predict $\underline{y}(t)$ as a function of $\underline{x}(t)$. A statistician would say that RHW-backpropagation is a special case of nonlinear least squares[6], a very well-studied method, with the restriction that the model is only allowed to use certain functional forms (sigmoids). Some argue that this arbitrary restriction "frees us" from the old econometricians' problem of figuring out what functional form to use; however, it would be more accurate to say that this otherwise arbitrary restriction can be convenient when coping with systems so complex that we have little apriori knowledge about which function to choose. My version of backpropagation allows the use of any differentiable function, and provides low-cost derivatives which can be input to methods more efficient than steepest descent[6].

In order to learn useful features, without enforcing a set of arbitrary targets $\underline{y}(t)$, many researchers have used a double-layer architecture which we may symbolize as $\underline{x}(t) \rightarrow \underline{R}(t) \rightarrow \hat{\underline{x}}(t)$. In other words, they set $y_j(t)$ in equation 2 to be $x_j(t)$ itself, so that the inputs and targets are the same. In the simplest case, they set N=m+k, where k is much smaller than m, and fix W_{ij} to be zero whenever i N and j m; in other words, the calculated components of net form two layers, in which the upper layer (which predicts $\underline{x}(t)$) is a function solely of the lower layer, which in turn calculates an intermediate vector $\underline{R}(t)$ (made up of net_{m+1} through net_{m+k}), which hopefully represents condensed features of the original inputs. There is an analogy here to factor analysis -- a very common statistical method -- but the neural net approach is nonlinear while factor analysis is not. A simple generalization of this approach would be to lengthen $\underline{y}$ to include both $\underline{x}$ and some classifications of interest. Because this architecture is a special case of equations 1 and 2, the RHW formulation of backpropagation can be applied to it directly.

For dynamical systems -- as in speech recognition, time-varying imagery or robotics -- I would recommend instead a triple-layer architecture, which I would symbolize (in minimal form) as $\underline{x}(t)$ and $\underline{R}(t-1) \rightarrow \underline{R}(t) \rightarrow \hat{\underline{R}}(t+1)$ and $\hat{\underline{x}}(t)$. This is the same as the double-layer architecture, except that the target vector

is lengthened to include both $\underline{x}(t)$ and $\underline{R}(t+1)$, and $\underline{R}(t-1)$ is available as an input at time t. (Thus equation 2 would have j run from 1 to n=m+k.) In this case, the vector $\underline{R}(t)$ need not be less than $\underline{x}$ in dimension, because it may represent "hidden" or "filtered" variables based on earlier time-periods which are not observable as functions of $\underline{x}(t)$. As a practical matter, of course, it would probably be best to build up the dimensionality of $\underline{R}$ in an incremental way, by first learning a few key components and then adding a few more, etc. Features derived in this way are more likely to represent basic, dynamic invariants or underlying features of the dynamic system.

This triple-layer architecture is a slight rephrasing of the "3-Net Architecture" discussed in a recent paper[7] which evaluates what has been learned from extensive studies of statistical theory and practical applications aimed at the problems of forecasting and the causal modelling of dynamic systems. This neural architecture (first proposed in 1974[1] and further discussed in 1977[8]) incorporates key ideas from time-series analysis and psychology, which have been tested and proven many times over. Simulation studies and social science forecasting studies have shown that some versions of this architecture lead to far less error, in forecasting dynamic systems over time, than does a naive architecture which takes $\underline{x}(t)$ as input and $\underline{x}(t+1)$ as targets. To achieve maximum improvement, however, equation 2 must be replaced by a weighted sum of error like:

$$\text{Error} = \sum_{t=1}^{T} \sum_{j=1}^{n} \tfrac{1}{2}\, \text{Weight}_j * (y_j(t) - s(\text{net}_{N+j}(t))^2 \qquad (9)$$

Procedures for choosing Weight have been developed and tested for social science applications, but for true sigmoid networks[7] there is a lot of room for further research and experimentation.

The three-net architecture is not a special case of equations 1 and 2. Therefore, we need to spell out a few more details to specify how to implement it. First, we must write out the equations of the feedforwards system implied by our discussion above. To begin with, we add the equation:

$$\text{net}_{m-k+i}(t) = \text{net}_{m+h+i}(t-1), \qquad i = 1 \text{ to } k \qquad (12a)$$

where the first m-k components of the network represent the $\underline{x}(t)$ inputs, the next k represent the vector $\underline{R}(t-1)$ used as an input, and equation 12 copies over the values of $\underline{R}(t-1)$ from where it was initially calculated (assuming h hidden units between the inputs and the calculation of $\underline{R}(t)$). We combine this equation with equation 1, under the understanding that W_{ij} will be zero for i greater than m+h+k and j less than m+h (although this restriction can be weakened[7]). Finally, we add this to a modification of equation 9:

$$\text{Error} = \tfrac{1}{2} \sum_{t=1}^{T-1} \sum_{j=1}^{k} \text{Weight}_j^{R} * (s(\text{net}_{m+h+j}(t+1)) - s(\text{net}_{N+m-k+j}(t))^2 + \tfrac{1}{2} \sum_{t=1}^{T} \sum_{j=1}^{m-k} \text{Weight}_j^{x} * (x_j(t) - s(\text{net}_{N+j}(t))^2 \qquad (12b)$$

Now that we have specified the equations of the feedforwards system, it is a straightforwards exercise to apply equation 5 to the system composed of equations 12a, 1, and 12b to generate the backwards equations to calculate all the derivatives. To do this, we proceed exactly as we did with equations 1 and 2 to

derive 6 through 8. (See [3,4] for very detailed illustrations and explanations of this procedure, including time-lags, based on practical applications at DOE.) The resulting recurrence equations must be applied backwards in time and backwards across neurons, both, but the required calculations cost about the same as it does to run the original system in forwards time, exactly as with conventional backpropagation[9]. (See the Appendix for details of this example.)

In actuality, I used exactly this kind of backpropagation through a recurrent system to minimize least square error in my very first applications test of backpropagation in 1974[1,8]. This application was implemented (and published/documented) in an MIT version of the Time-Series Processor (TSP) software package. Our earlier work also includes derivative propagation through a doubly recurrent system[9], including both lagged recurrence and simultaneous-time recurrence (which can be applied to continuous dynamic systems, ala Grossberg and Hopfield, embedded within a time-modulated system). In principle, equation 12b assumes a normal error distribution, and should be adjusted slightly to account for the zero-to-one range of the sigmoid function.

This kind of derivative calculation fits very nicely with batch learning, and can even be used with pattern learning (in backwards time), but forwards-time real-time systems like the brain cannot perform these calculations exactly. Still, there are ways to approximate these calculations in a real-time system, which have remarkable biological parallels[8]. Even though backpropagation is not yet proven in the brain, it is not yet disproven either. For one thing, one cannot yet rule out Freud's theory that "psychic energy", a chemical backflow (which could implement backpropagation, if fast enough for some chemical species) drives the adaptation of synapses. Furthermore, recent studies have shown that components of the microskeleton in many cell types can carry mechanical and electromechanical information forwards and backwards, as fast as would be needed. In general, our framework leads to many testable hypotheses about the brain, elaborated on in the papers cited here.

CONTENT-ADDRESSABLE MEMORY AND CONVERGENCE

Both for the sake of accuracy and for the sake of better real-time convergence[10], more research is needed into ways of blending backpropagation and content-addressable memory as formulated by Kohonen[10]. (Other formulations -- especially continuous-time formulations -- are important to analog implementation, but we also need a better understanding of what to implement.)

A purist statistician would argue that least squares is the correct way to handle supervised learning problems, including those of the prior section, because it converges to the model which has the maximum likelihood of being true, which in turn should give the best forecasts. "Likelihood," by definition, treats all models as equally likely apriori. Using correlation coefficients (like the usual $W=X^TY$ "approximation" of Kohonen) instead of regression coefficients gives rapid convergence to an inaccurate model, unless we can truly guarantee a linear relation from features to outputs and a true lack of correlation between all features in the universe being sampled from (which is highly unlikely when features are nonlinear functions of each other).

Nevertheless, different models are not equally probable apriori in practice. R. Solomonoff and Abu-Mostafa have reminded us that there is no generic adaptation procedure which is right for all conceivable environments (either for maximum accuracy or for fast convergence), and that we must invoke Occam's Razor and prior probability distributions to justify any kind of adaptation or intelligence, mechanical or human. Based on similar considerations, and extensive empirical tests, mainstream statisticians like Dempster and Efron now

advocate "ridge regression," which is easily generalized to nets by modifying equation 2 (or 9 or 12b) to:

$$E' = \text{Error} + K \sum W_{ij}^2 \quad \text{or Error} + \sum K_i W_{ij}^2, \tag{13}$$

and otherwise continuing as in the preceding section. When steepest descent is applied with a fixed learning rate, and K is small, this yields the well-known "decay term" which has often been tried -- to little effect -- as a way of speeding up the convergence of backpropagation. Nevertheless, when the number of inputs is large relative to the number of patterns, ridge regression requires that we use much larger values of K, for the sake of accuracy, not for the sake of convergence. As K goes to infinity, the results approach Kohonen's method (to within a murky scalar factor). In tests with nets on a PC at my home, I have found that large values of K tend to stabilize the weights relative to changes in the pattern set; however, it is essential to add a kind of global weight to each neuron (i.e., set output to s(global_weight*net)), where the global weight is kept out of equation 13, to get the scalar factors right. Benefits to convergence have been modest so far, but I have yet to test alternative algorithms[7,10] which approach Kohonen's solution procedure as K goes to infinity. (I have also developed a continuous-time version[10] which is crude and approximate, but converges in one pass without using backpropagation per se.) In general, the human ability to learn from a single instance and to assimilate memories later on into a more rule-based causal understanding underlines the importance and potential of this kind of blending.

Ultimately, we will have to remember that "supervised learning" is really two different classes of problems (based on different probability distributions). In one case, discussed above, we are mainly trying to forecast new situations, and - even with equation 13 added - the backpropagation approach seems best. In the other, we only try to recreate past situations. An ideal system would combine both capabilities, so that the predictive network can be adapted both to current experience and to remembered experience. Parallel adaptation to and use of multiple experiences may even be possible to some degree, especially in a deep sleep kind of mode. If so, then human adaptation may be more like batch learning than one might have guessed at first.

In DOE applications of steepest descent and related methods, we have generally achieved convergence in 10 to 40 iterations, using batch adaptation with an adaptive learning rate and carefully-designed "scaling factors"[3,4] (which yield different learning rates for different weights). In a 1981 review for the EIA Quality Assurance Division[12], I suggested a few approaches for using adaptation to derive the scaling factors, but I have not compared these approaches to others suggested by Sutton, Parker, and my Ph.D. thesis[1].

In recent papers on building intelligent systems[2,9,13,14], I have stressed the importance of borrowing methods from numerical analysis (and cited [12]) to adapt neural networks, which are really just a special case of generalized functional networks. Since Watrous' paper here last year, the neural net community is well aware that any method to minimize a complex function can be applied to neural networks.

What kinds of numerical methods are most useful with large-scale, sparse problems -- the kind of problems represented by neural nets? Dennis and Schnabel[6] cite papers on conjugate gradients as the most promising approach to such problems, including a paper by Shanno[15]. Charles Mylander of the U.S.Naval Academy, in a review of our use of backpropagation at DOE[3,4], also recommended that we look more closely at conjugate gradients. Shanno describes basic methods, like Fletcher-Reeves and Polak-Ribiere, which are basically as

fast and cheap as the similar "momentum" method of RHW, but far more effective (at least in minimizing quadratics and solving linear equations). Shanno proposes other methods, also O(n) in cost, which he claims are far more robust in dealing with complex, nonlinear problems.

David Parker tells us that the neural net community is also interested in full Newton's methods but discouraged by the high storage costs. However, my previous review[12] discussed a method by Bank and Rose[16] which allows a full Newton's method at O(n) storage cost. Given known methods to solve linear equations $g=Hx$, their method simply requires that we can calculate Hx economically for an arbitrary vector x, where H is the Hessian (second derivative matrix) of Error with respect to the weights. In 1979, I pointed out that such vectors can be calculated at O(n) cost[11,2]. In this approach, we define:

$$z = \sum F\_W_{ij} * x_{ij}, \tag{14}$$

and treat the combination of equations 1, 6, 7, 8, and 14 as a single feedforwards system of equations to calculate the target z. We can apply the chain rule for ordered derivatives to this entire system to calculate the derivatives of z with respect to the weights, derivatives which equal the contents of Hv. (See Appendix for a few more details.) Numerical analysts[6] have developed methods to deal with the trust region problems of Newton's methods, problems which are also present, but more concealed, with the simpler methods.

All of these methods have been exhaustively tested over conventional numerical problems (i.e. batch problems, such that batch learning might outperform pattern learning sometimes until we adapt these methods), but there are a variety of ways to adapt them to pattern learning, to approximate them, combine them, and cut corners as well. For example, adaptive scaling factors could be incorporated in the "preconditioning" matrices of various conjugate gradient methods, and related to standard errors as defined by statisticians. Pattern learning has a relation to both SOR methods and preconditioning.

Everything I have seen so far on these lines is quite promising, but there is a lot more research to be done.

REINFORCEMENT LEARNING, OPTIMIZATION, ROBOTS AND BRAINS

One of the key applications of AI is in designing systems which do something -- take overt action -- like robots and brains. At last year's conference, Kawato et al gave an impressive paper, showing how a biological analysis by Uno involving optimization over time leads to an adaptive robot which already compares well with anything in the mainstream robotics literature. John J. Craig of Stanford has recently written books which represent the best of the mainstream work in robotics; from these books, it seems clear that robotics are a major near-term target of opportunity for neural networks, if we can combine straightforward causal modelling (as discussed in the previous two sections) along with capabilities for optimization over time. Mathematically, the challenge is to take action so as to minimize a cost function or, equivalently, to maximize a utility function, over time. As an ethical matter, however, we should remember that neural networks are probably fully capable of reproducing the worst nightmares of science fiction, if we are not careful.

In a similar vein, cognitive psychologists have shown more interest in "reinforcement learning" than in classical supervised learning or unsupervised learning, because neither of the latter two is really plausible as a model of natural intelligence. In the simplest versions of reinforcement learning, the

system inputs a measure of reinforcement (or utility U) which it first tries to predict as a function of other variables, including both sensory inputs $\underline{x}(t)$ and motor control variables $\underline{u}(t)$. (This is a straightforward supervised learning problem, where U(t) is the target and $\underline{x}(t)$ and $\underline{u}(t)$ are the inputs.) A secondary network or action network is also set up, which inputs $\underline{x}(t)$ and outputs $\underline{u}(t)$. To adapt the parameters of the action network, one focuses attention on the entire two-component network going from $\underline{x}(t)$ to $\underline{u}(t)$ and from there to a prediction of U(t); one uses backpropagation through the entire network to calculate the derivatives of $\hat{U}(t)$ with respect to the weights in the action network, and one adapts them accordingly. Barto, Sutton and Anderson, as well as myself [14,15], have emphasized that this version leaves out the crucial problem of maximizing over time; we have also discussed at length the idea of utility maximization as a paradigm for human intelligence, both from a biological[13,17] and humanistic[14,18] point of view.

Some computer scientists have questioned our continued tendency to go back to biological and psychological analogues and citations. However, this kind of cross-fertilization and reverse engineering is what got this field started, and it would be a serious mistake to abandon it now. Overspecialization has already led to too much reinventing of the wheel already. In fact, one might even argue that the greatest long-term benefit of this research - including the purest mathematical work - would be in helping humans to understand themselves a little better.

In any event, the problem of optimization over time is relatively simple when we have a predictive model and do not account for random factors. We have had good results with backpropagation here in a practical application, involving the main model used by the Energy Information Administration in its official forecasts[3,4]. The key idea was to define the feedforwards system as the model itself (as if exactly true) plus a utility function, across time. We then used backpropagation to calculate the derivatives of total utility across time with respect to the action variables at all times. We adjusted the action variables in accord with these derivatives, to arrive at an optimal schedule of action. Based on this approach, we are now looking at a larger-scale optimization problem which includes uncertain outcomes whose probabilities are projected by a fuzzy inference system; because these probabilities are continuous functions of each other and of continuous input variables, we can still apply equation 5 (or its generalization to simultaneous systems[9]) to finding the optimal actions. In a true stochastic problem, a more complex solution is required[13,14], based on an effort to approximate dynamic programming.

I have proposed three major designs, of varying complexity, for the stochastic optimization problem[2,14]. Much more research is needed to test and understand these designs, though the success of related methods used by Barto, Sutton and Anderson leads me to expect good results. All three designs require that we adapt a "strategic assessment" network, analogous to the "adaptive critic" of Barto, Sutton and Anderson. In the most elementary approach, this network would input both $\underline{x}(t)$ and $\underline{u}(t)$, and output J(t), a measure of the overall strategic benefit of the current situation. It would use supervised learning to adapt this network, setting the target for J(t) to be J(t+1)+U(t+1)-U0, where J(t+1) is treated as a constant when we adapt the network for time t and where U0 is a constant threshold term also to be adapted. (U0 is important to prevent divergence in situations - like those of living organisms - where there is no clear termination time, and reinforcement is received throughout the process.) Then, to adapt the weights in the action network, we would consider the entire two-layer feedforwards network from $\underline{x}(t)$ to $\underline{u}(t)$ and from $\underline{x}(t)$ and $\underline{u}(t)$ to J(t);

we would use backpropagation through this entire network to calculate the derivatives of J with respect to the weights in the action network, and adapt those weights in accord with these derivatives. This scheme can be modified to exploit the availability of a network to make predictions (or stochastic simulations), like those discussed in the previous two sections; this would enhance the system's capabilities, especially in dealing with combinations of events which have never occurred in the past, and it would reduce the value of allowing $\underline{u}(t)$ as an input to the J(t) network.

In conventional "adaptive" robotics, robots often adapt by virtue of the fact that $\underline{u}(t)$ is a function of $\underline{x}(t)$. The approach here combines that kind of adaptation with adaptation of the weights which give that function.

The more advanced designs (which are much more plausible as models of the human brain[13]) require a prediction network, and make full use of the cause-and-effect relations implied by that network. In the main method, GDHP, we try to minimize the sum of the squares of the derivatives of $J(t+1)+U(t+1)-J(t)$ with respect to the inputs of the J network, added up over the inputs. The derivatives of that error measure - which itself includes derivatives - require the calculation of second derivatives, calculated by the procedure given in the previous section (discussed in [2,13], along with a variety of details on these methods and comparisons with experimental evidence from neurophysiology).

REFERENCES

1.P.Werbos,Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Ph.D. thesis, Harvard U. Committee on Applied Mathematics, Nov. 1974.

2. P.Werbos,"Applications of Advances in Nonlinear Sensitivity Analysis" in R.Drenick and F.Kozin eds., Systems Modelong and Optimization: Proceedings of the International Federation for Information Processing. Springer-Verlag, 1982.

3. P.Werbos,"Documentation of the Gas Analysis Spreadsheet (GAS) As Used in the Annual Energy Outlook 1987". National Energy Information Center (NEIC), EIA, Department of Energy, Washington D.C. 20585 (202-586-8800), 1988.
(Should be free of charge; otherwise contact author.)

4. P.Werbos,"Maximizing Long-Term Gas Industry Profits in Two Minutes in Lotus Using Neural Network Methods", draft, 1988.

5. S.Foote and J.Morrison,"Extrathalamic Modulation of Cortical Function",Annual Review of Neuroscience, 1987.

6. J.Dennis and R.Schnabel,Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Prentice-Hall, 1983.

7.P.Werbos,"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, Vol.1. IEEE Catalog No. 87CH2503-1. IEEE, 1987.

8. P.Werbos,"Advanced Forecasting Methods for Global Crisis Warning and Models of Intelligence",General Systems Yearbook 1977.

9. P.Werbos,"AGeneralization of Backpropagation to Recurrent Networks, With a Gas Market Application and Neural Implications", submitted to Neural Networks, 1987,

and revised and resubmitted per instructions in 1988.

10.P.Werbos,"Backpropagation Versus Content Addressable Memory: Applications, Evaluation and Synthesis", draft.

11.P.Werbos,"Changes in Global Policy Analysis Procedures Suggested by New Methods of Optimization",Policy Analysis and Information Systems.Vol.3,No.1,1979.

12.P.Werbos,"Solving and Optimizing Complex Systems: Lessons from the EIA Long-Term Model", in Energy Models and Studies, B.Lev Ed. North Holland, 1983.

13. P.Werbos,"Building and Understanding Adaptive Systems: A Statistical/Numerical Approach to Factory Automation and Brain Research",IEEE Transactions on Systems, Man and Cybernetics, Jan.-Feb. 1987.

14. P.Werbos,"Generalized Information Requirements of Intelligent Decision-Making Systems", in SUGI 11 Proceedings. Cary, N.C.: SAS Institute, 1986. (A revised version, available from the author, is easier to read and contains more discussion of psychology.)

15.D.Shanno,"Conjugate-Gradient Methods With Inexact Searches",Mathematics of Operations Research, Vol. 3, August 1978.

16.R.Bank and D.Rose,"Parameter Selection for Newton-Like Methods Applicable to Nonlinear Partial Differential Equations",SIAM J. Num. An., Dec. 1980.

17. P.Werbos,"The Elements of Intelligence",Cybernetica, No. 3, 1983. (Namur)

18. P.Werbos,"Rational Approaches to Identifying Policy Objectives" in Planning in A Risky Environment: A Handbook of Energy/Economy Modeling, J.Weyant and T.Kuczmowski eds., Pergamon, forthcoming 1988.

APPENDIX: APPLICATION OF EQUATION 5 TO EXAMPLES GIVEN IN THIS PAPER

For the 3-layer architecture in dynamic modelling, we use equations 12a, 1, and 12b to define the feedforwards system to be differentiated. The last quantity to be calculated in that system, prior to the target variable (Error), is $net_i(t)$ for $i > N+m-k$, which appears on the right-hand side of equation 12b for t less than T. Differentiating equation 12b with respect to such $net_i(t)$, and plugging into equation 5, we derive an equation for such i:

$$F\_net_{N+m-k+i}(t) = Weight_i^R \, (s(net_{M+m-k+i}(t) - s(net_{m+h+i}(t+1))s'(net_{M+m-k+i}(t)) \quad (15)$$

The next to last set of quantities are $net_i(t)$ for $N+m-k \geq i > N$, which appears on the right only in equation 12, yielding the following for all t for such i:

$$F\_net_{N+i}(t) = Weight_i^X * (s(net_{N+i}(t) - x_i(t)) * s'(net_{N+i}(t)) \quad (16)$$

With $net_i(t)$, for $N \geq i > m$, we still get equation 7, except for i between m+h+1 and m+h+k and t less than T; for such units, we add the term $F\_net_{i-h-k}(t+1)$, which results from differentiating equation 12a. Finally, equation 8 still applies, as before.

For the second derivative calculation described above, we differentiate the system made up of equation 1, 6, 7, 8 and 14, with "F_" changed to "G_" to avoid ambiguity. The last quantities calculated in the system are G_W_{ij}, used in equation 14; when we plug into equation 5, we get:

$$F\_G\_W_{ij} = x_{ij} \tag{17}$$

The next to last quantities are $G\_net_i(t)$ for all i, which appear on the right both in equations 7 and 8. Applying equation 5, we get:

$$F\_G\_net_i(t) = \sum_{i>j} F\_G\_W_{ij}\, s(net_j(t)) + \sum_{i>j} F\_G\_net_j(t) * W_{ij} s'(net_j(t)) \tag{18}$$

Bear in mind that the leftmost summation here will often be zero when there is no weight connecting two neurons. Also note that we have to use this equation in the forwards direction, invoking it first for i=m+1, then m+2, and so on. (We do not need to calculate $F\_G\_net_i$ for i less than m, because we did not calculate G_net_i for such i in the original system of equations 1, 6, 7, 8, and 14.)

The equation to calculate $F\_net_i(t)$ is more complex, because $net_i(t)$ appears in almost all the equations:

$$\begin{aligned} F\_net_i(t) = {} & s'(net_i(t)) \sum_{i<j} (F\_G\_W_{ji} * F\_net_j(t) + F\_net_j(t) * W_{ji}) \\ & + s''(net_i(t)) \sum_{i<j} F\_G\_net_i(t) * F\_net_j(t) * W_{ji} \\ & + F\_G\_net_i(t) * ((s'(net_i(t)))^2 - e_{i-N}(t) * s''(net_i(t)), \end{aligned}$$

where the last term (lowest line) applies only for $i \leq N$, and where $e_{i-N}(t)$ refers to $s(net_i(t)) - y_{i-N}(t)$. This equation is calculated for decreasing i, as was done earlier for equations 6 and 7 in the original system.

Finally, to calculate the F_W_{ij} in this system (which are the components of the vector $H\underline{x}$), we get:

$$F\_W_{ij} = \sum_t \left(F\_G\_net_j(t) * F\_net_i(t) * s'(net_j(t)) + F\_net_i(t) * s(net_j(t)) \right),$$

where the leftmost term is taken to be zero for $j \leq m$.

These calculations have been checked, and the three-net equations have been tested in many other circumstances (using similar but more complex equations); however, while equation 5 in general has been thoroughly reviewed and verified over many years and many applications, I cannot completely rule out the possibility of a typographical error in this paper.