Figure 7

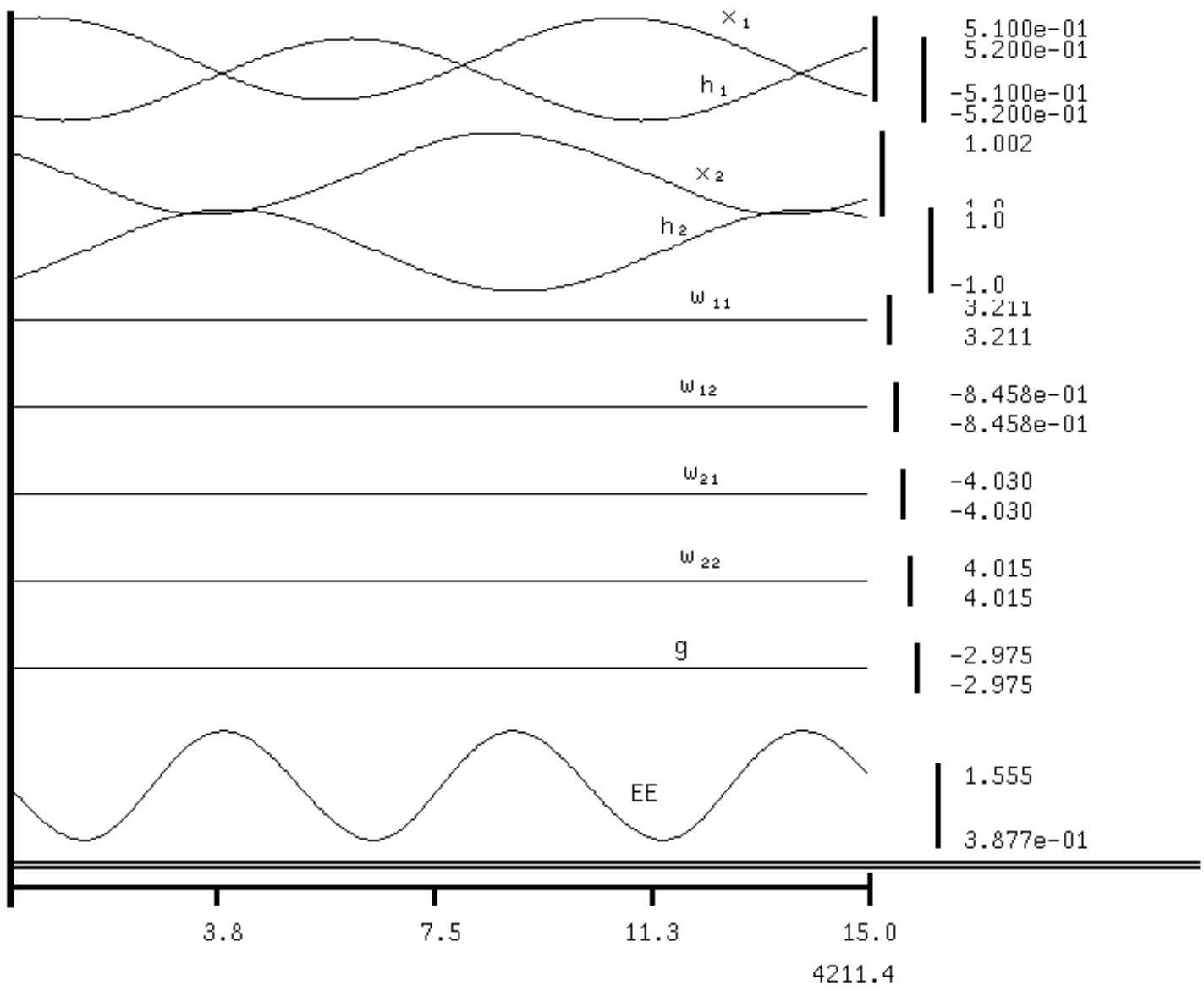

x1
h1
x2
h2
w11
w12
w21
w22
g
EE
5.100e-01
5.200e-01
-5.100e-01
-5.200e-01
1.002
1.0
-1.0
3.211
3.211
-8.458e-01
-8.458e-01
-4.030
-4.030
4.015
4.015
-2.975
-2.975
1.555
3.877e-01
3.8
7.5
11.3
15.0
4211.4

Figure 6

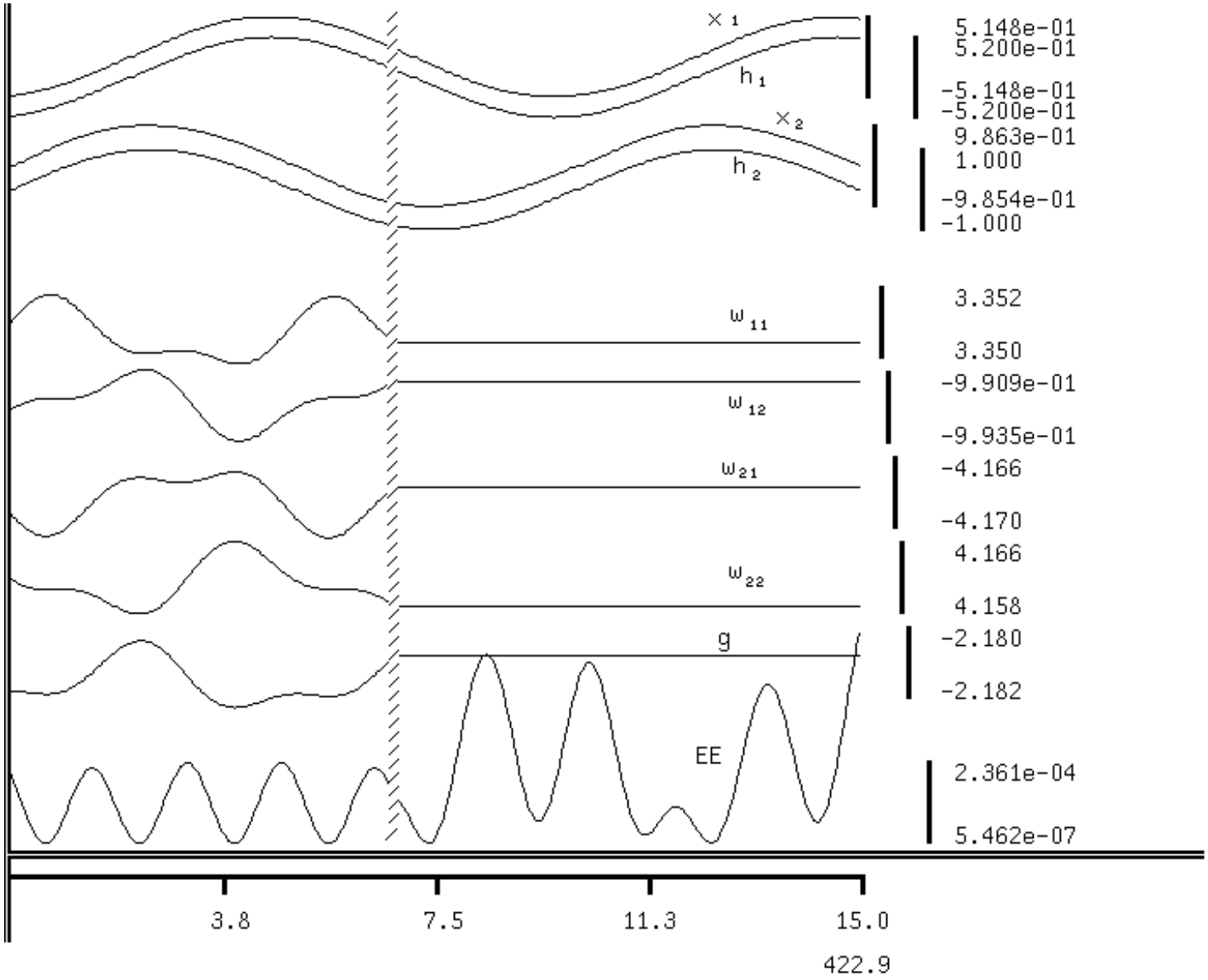

x 1
h 1
x 2
h 2
w 11
w 12
w 21
w 22
g
EE
5.148e-01
5.200e-01
-5.148e-01
-5.200e-01
9.863e-01
1.000
-9.854e-01
-1.000
3.352
3.350
-9.909e-01
-9.935e-01
-4.166
-4.170
4.166
4.158
-2.180
-2.182
2.361e-04
5.462e-07
3.8
7.5
11.3
15.0
422.9

Figure 5

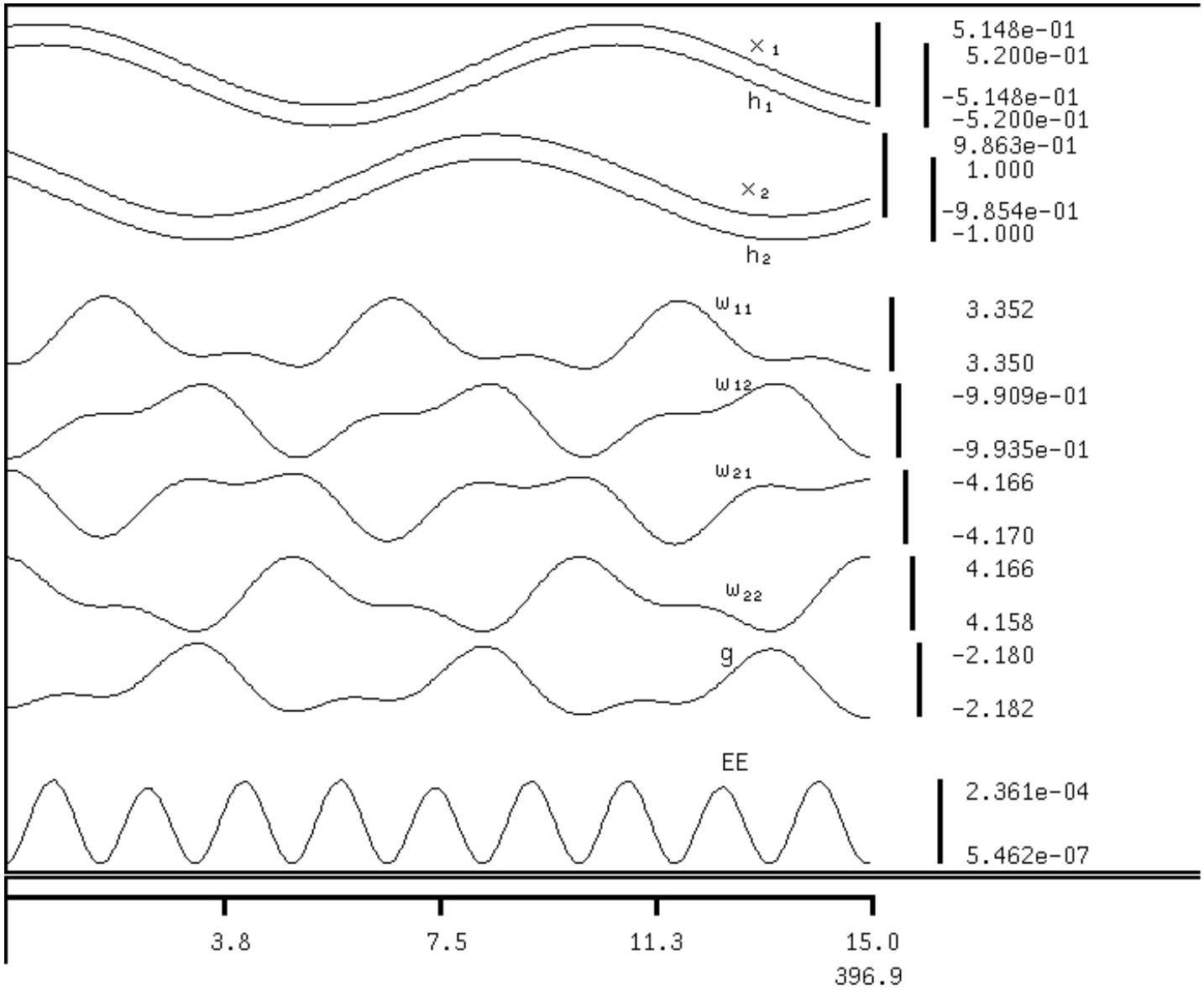

x1
h1
x2
h2
w11
w12
w21
w22
g
EE
5.148e-01
5.200e-01
-5.148e-01
-5.200e-01
9.863e-01
1.000
-9.854e-01
-1.000
3.352
3.350
-9.909e-01
-9.935e-01
-4.166
-4.170
4.166
4.158
-2.180
-2.182
2.361e-04
5.462e-07
3.8
7.5
11.3
15.0
396.9

Figure 4

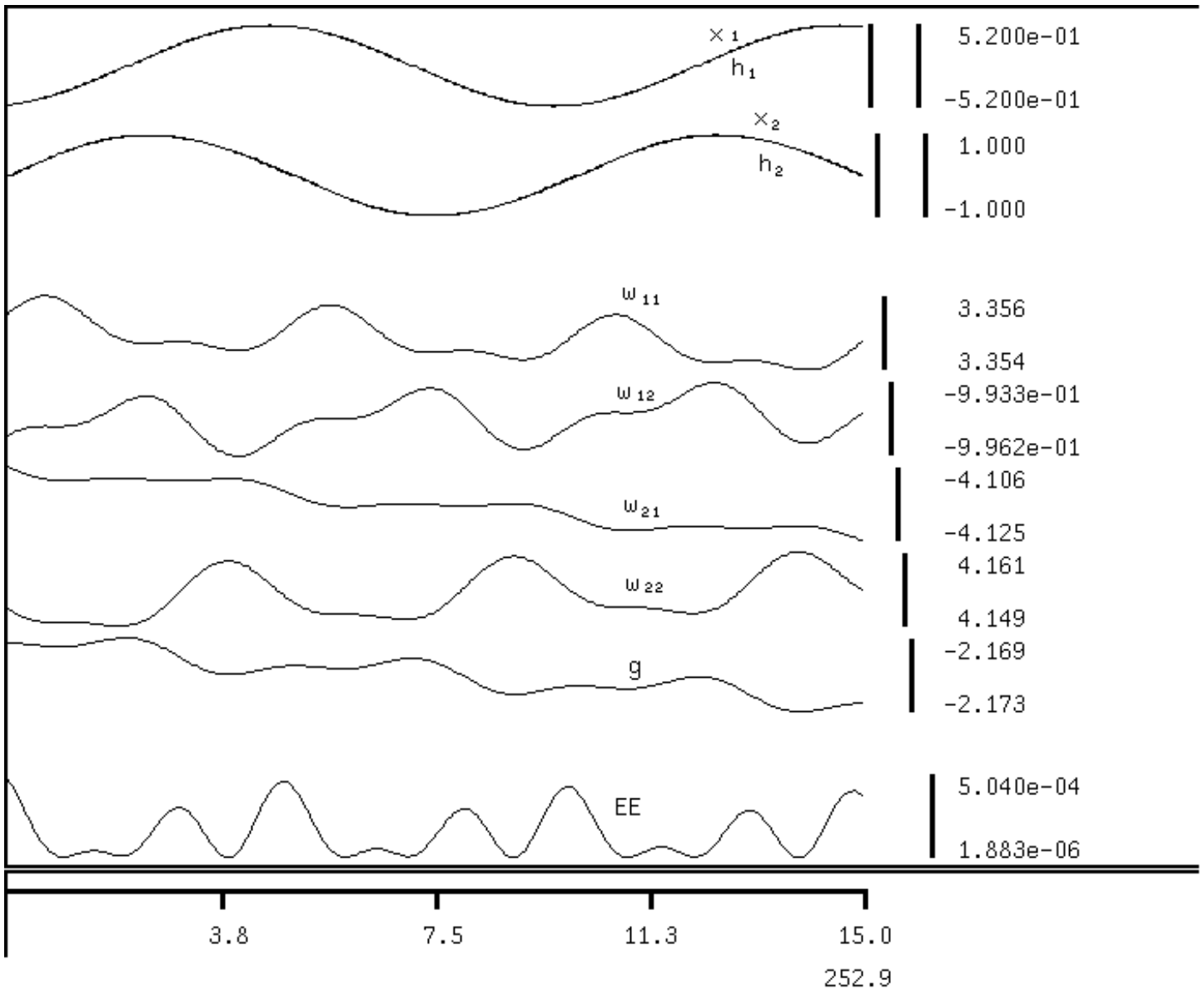

x1
h1
5.200e-01
-5.200e-01
x2
h2
1.000
-1.000
w11
3.356
3.354
w12
-9.933e-01
-9.962e-01
w21
-4.106
-4.125
w22
4.161
4.149
g
-2.169
-2.173
EE
5.040e-04
1.883e-06
3.8
7.5
11.3
15.0
252.9

Figure 3

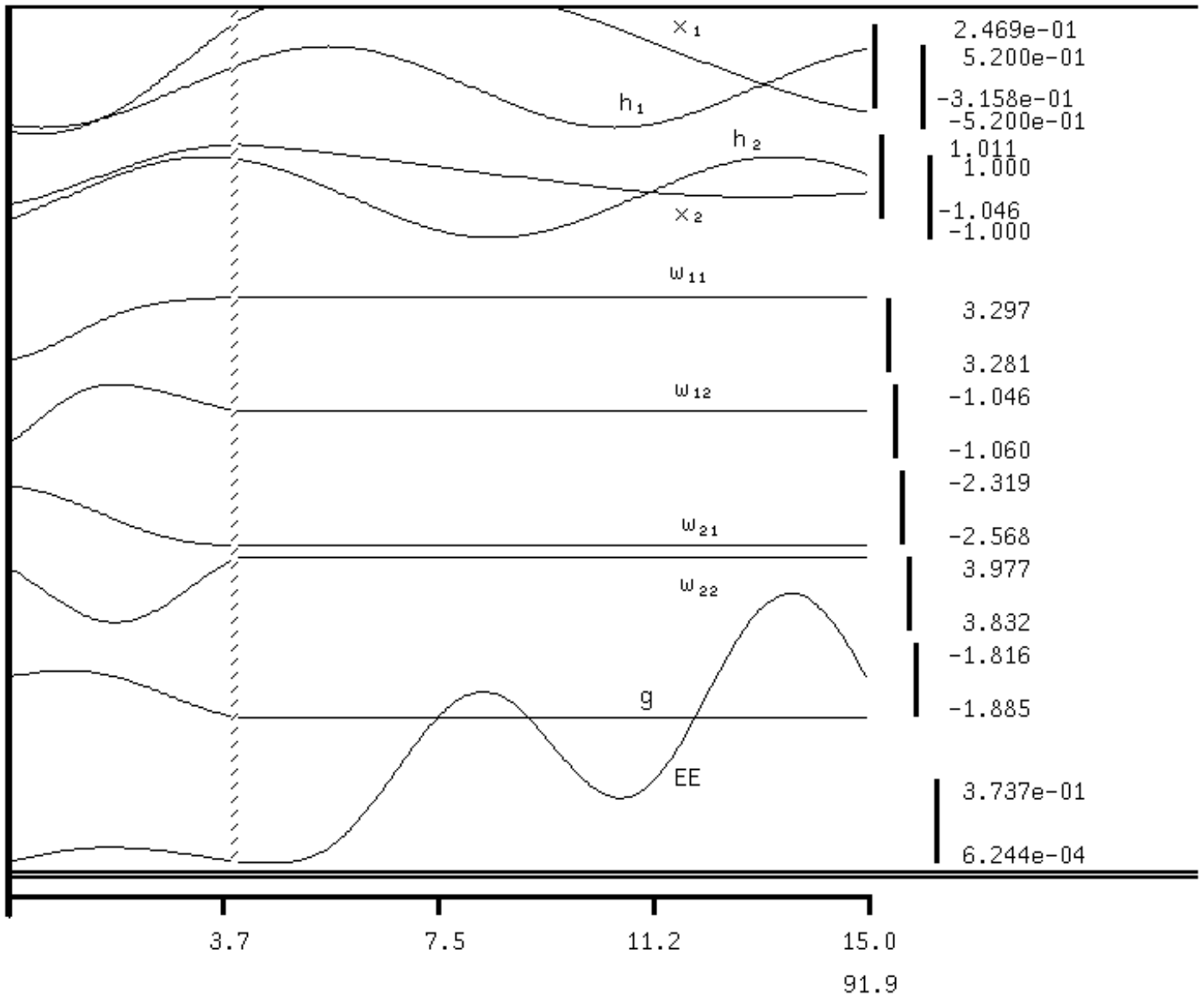
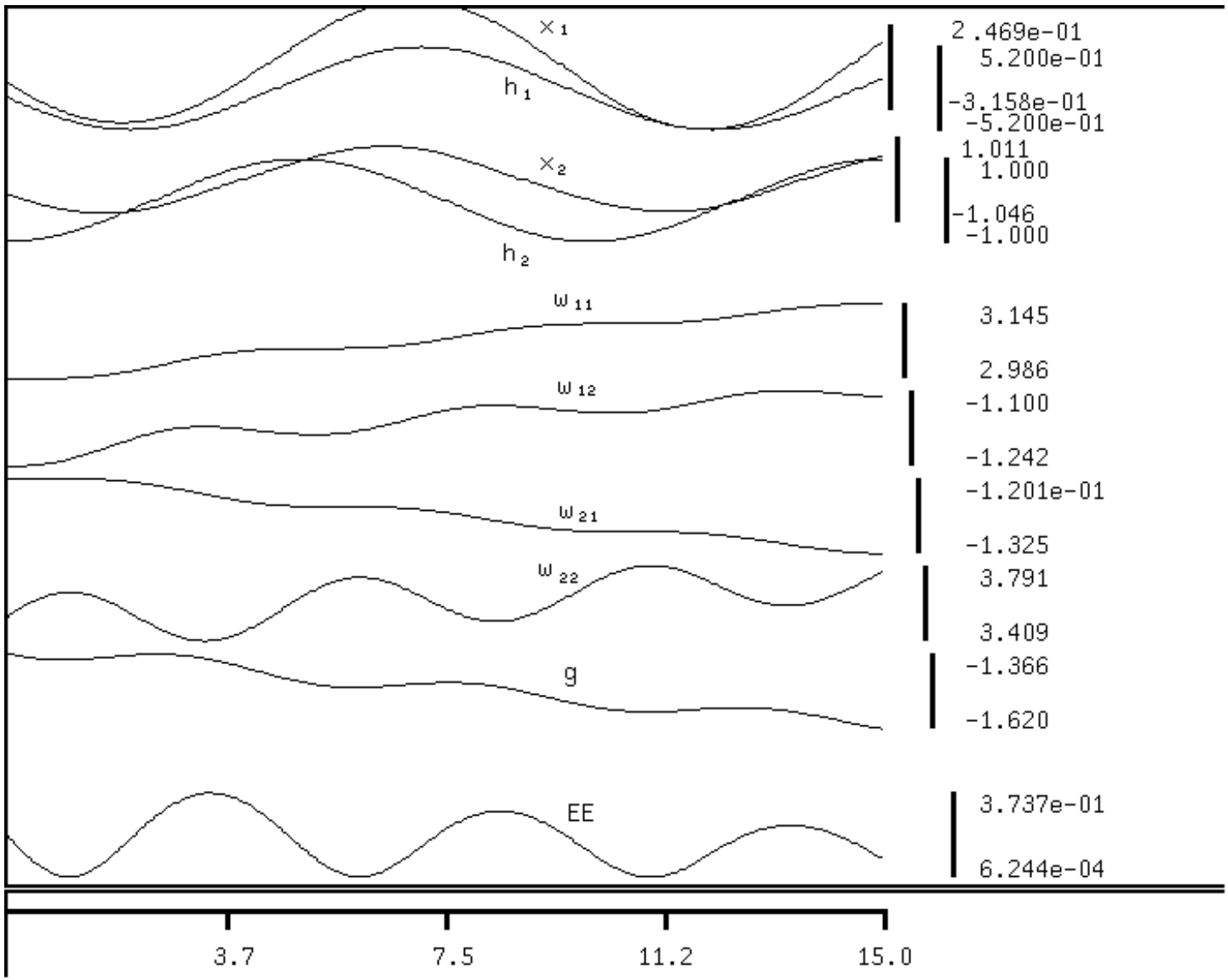
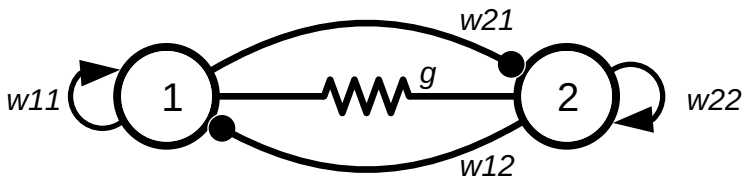
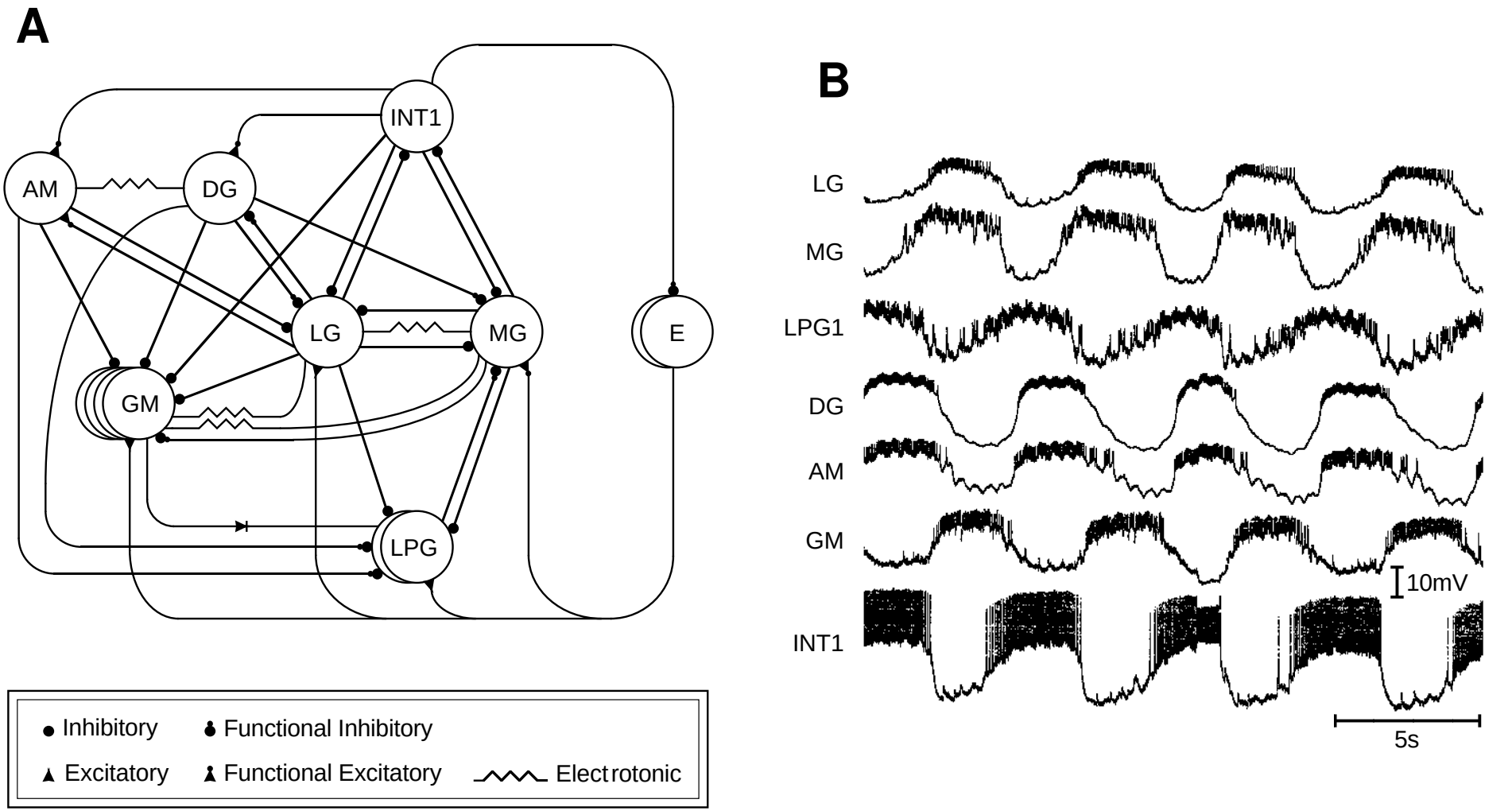

x1
2.469e-01
5.200e-01
h1
-3.158e-01
-5.200e-01
h2
1.011
1.000
x2
-1.046
-1.000
w11
3.297
3.281
w12
-1.046
-1.060
-2.319
w21
-2.568
3.977
w22
3.832
-1.816
g
-1.885
EE
3.737e-01
6.244e-04
3.7
7.5
11.2
15.0
91.9

Figure 2


x1
h1
x2
h2
w11
w12
w21
w22
g
EE
2.469e-01
5.200e-01
-3.158e-01
-5.200e-01
1.011
1.000
-1.046
-1.000
3.145
2.986
-1.100
-1.242
-1.201e-01
-1.325
3.791
3.409
-1.366
-1.620
3.737e-01
6.244e-04
3.7
7.5
11.2
15.0
40.4


w21
w11
g
1
2
w22
w12


A
INT1
AM
DG
LG
MG
E
GM
LPG
Inhibitory
Functional Inhibitory
Excitatory
Functional Excitatory
Electrotonic
B
LG
MG
LPG1
DG
AM
GM
INT1
10mV
5s

free can be easily compared with the desired behaviour. The network continues to oscillate, in contrast to the behaviour of figure 2. The error term EE shows a slight increase and the uniform nature of the six peaks begins to be lost. The error term is being computed and displayed for comparison purposes only.

Figure 7. On a different run, a different set of parameter values was found to produce the target outputs. After learning was complete the network was allowed to run free for another 3000 model time units, or for 300 complete cycles. The state variables drifted 180° out of phase with the target outputs due to the learnt period not being exactly identical with the target output period. The learnt period is within 1/6 of 1 percent of the target output period. If the learning is continued while the learning rate η is made very small, the accuracy of the learnt signal increases. The error signal is of course very large.

the parameters are adjusted by back-propagation.

Figure 3. An illustration of the result of letting the network run free before learning is complete. Training and forcing were turned off at the dotted line; therefore the parameter values w_{11}, w_{12}, w_{21}, w_{22}, and g become constant thereafter. The network is not yet able to oscillate, as can be seen by the values for x_1 and x_2 which are decaying towards a fixed point. The error term shows the beginning of large oscillations as a result of the state variables decaying to constant values.

Figure 4. The output when learning is nearly complete. The traces for the pairs x_1, h_1, and x_2, h_2 have been superimposed to demonstrate how closely they agree. No difference is detectable between each pair. Their actual difference is summarized by the error signal EE at the bottom of the display. Note that the maximum value of the error signal is less than 0.0005. Note also that it is beginning to show the appearance of six peaks for every peak of the training signal, i.e. an oscillation with period 1/6 of the training signal period. When "six peaks" appear, this reliably means that the network will now oscillate on its own when allowed to run free. The five parameters are still changing, but only very slowly: note the very small amplitudes of the parameter oscillations. w_{11}, w_{21}, and g are decreasing, while w_{12} and w_{22} are increasing.

Figure 5. Model output when learning is complete. This is indicated by the uniform nature of the six peaks in the display of the error signal EE. The traces for the pairs x_1, h_1 and x_2, h_2, have been moved apart so that the agreement between each state variable and its target output is visible. Note the continued presence of very slight differences (1%) between the amplitudes of the state and forcing signals, and how the parameter values are approximately constant across the width of the display although in fact longer-range movement is still occurring. If learning is continued long enough the discrepancies in the amplitudes of the state variables and forcing signals reduces towards zero and the parameter values become completely constant. Time = 396.9 model time units, thus learning was complete after fewer than 40 complete cycles of the target outputs.

Figure 6. Learning and forcing were both turned off at the dotted line, although the target outputs are still displayed so that the performance of the network when running

Figure captions

Figure 1. 1A shows the network connections of the gastric mill central pattern generator in the lobster. Solid triangles - excitatory synaptic connections; solid circles - inhibitory synaptic connections; resistor symbols - gap junctions; connection from GM to LPG - rectifying gap junction (current flow one way only); thicker lines denote stronger connections.

1B shows the waveforms of the membrane potentials of each cell in the network. The four GM cells are electrically connected to each other and all produce the same waveform. Similarly for the two LPG cells. The spikes from each cell do not play a major role in pattern generation. The higher-frequency superimposed waveforms, particularly noticeable in the DG, AM, and INT1 output, are caused by inputs from another CPG in the same ganglion. Calibration bars - 1s, 10 mV.

Figure 2. The beginning of learning in the model specified by equations 8.1 (modified by restricting g to be positive), 8.2, 8.3, and 8.4. This model has the network diagram shown at bottom. Each vertical bar on the right hand side of the figure shows the vertical extent of a rectangular area of the computer screen set aside for one trace. Rectangular areas for different traces can overlap and each trace can be individually scaled to lie within the its rectangle. The upper and lower numbers calibrate the upper and lower edges of the rectangle of a trace. The time scale is along the bottom, and the lowest figure, at the right hand side, is the time of the right hand end of the traces since the model started running.

Traces h_1, h_2, show the target output and traces x_1, x_2 show the response of the forced system. x_1 is already almost in phase with h_1 but the amplitude of x_1 is only 0.56, roughly half the amplitude of h_1, which is 1.04. On the other hand, x_2, h_2, are out of phase but have approximately the same amplitude. The period of the target output is 10.0 model units. The five parameter traces, w_{11}, w_{12}, w_{21}, w_{22} and g show how the parameter values are being adjusted in oscillatory fashion by the learning algorithm. The trace labeled 'g' shows how the value of the restriction variable g^* is being adjusted by the algorithm. The actual value for the conductance g is given by $pos(g^*) = exp(0.1g^*)$, consequently the true numerical extents for the trace of g are 0.85 and 0.87. EE is the error term; note how it is decreasing rapidly in magnitude as

Free equations:

$$\tau_i \frac{\mathrm{d}x_i}{\mathrm{d}t} + x_i = \sum_{j=1}^{n} w_{ij}\, s(x_j) + \sum_{j=1, j\neq i}^{n} pos(g_{ij})(x_j - x_i) \qquad \text{for } 1 \leq i \leq n$$

$$\text{with } g_{ij} = g_{ji} \qquad \text{for all } (i,j), i \neq j.$$

Error term:

$$E = \frac{1}{2} \sum_{i=1}^{n} (x_i - h_i)^2.$$

Sensitivity coefficients:

$$z_{ij} = \frac{\partial x_i}{\partial w_{ij}}, \qquad \text{for } 1 \leq i, j \leq n,$$

and for each pair (i,j), $1 \leq i < j \leq n$, define

$$\gamma_{ij}^{i} = \frac{\partial x_i}{\partial g_{ij}}, \qquad \gamma_{ij}^{j} = \frac{\partial x_j}{\partial g_{ij}}.$$

Forced equations:

$$\tau_i \frac{\mathrm{d}x_i}{\mathrm{d}t} + x_i = \sum_{j=1}^{n} w_{ij}\, s(h_j) + \sum_{j=1, j\neq i}^{n} pos(g_{ij})(h_j - h_i), \qquad \text{for } 1 \leq i \leq n,$$

$$\text{with } g_{ij} = g_{ji} \text{ for all } (i,j), i \neq j.$$

Sensitivity equations:

$$\tau_i \frac{\mathrm{d}z_{ij}}{\mathrm{d}t} + z_{ij} = s(h_j) \qquad \text{for } 1 \leq i, j \leq n.$$

For each (i,j), $1 \leq i < j \leq n$, a pair of equations

$$\tau_i \frac{\mathrm{d}\gamma_{ij}^{i}}{\mathrm{d}t} + \gamma_{ij}^{i} = pos'(g_{ij})(h_j - h_i),$$

$$\tau_j \frac{\mathrm{d}\gamma_{ij}^{j}}{\mathrm{d}t} + \gamma_{ij}^{j} = pos'(g_{ij})(h_i - h_j).$$

Adjustment equations:

$$\frac{\mathrm{d}w_{ij}}{\mathrm{d}t} = -\eta(x_i - h_i) z_{ij} \qquad \text{for } 1 \leq i, j \leq n, \text{and}$$

$$\frac{\mathrm{d}g_{ij}}{\mathrm{d}t} = -\eta\left\{(x_i - h_i)\gamma_{ij}^{i} + (x_j - h_j)\gamma_{ij}^{j}\right\} \qquad \text{for } 1 \leq i < j \leq n.$$

Appendix 1. Implementation and system notes

We have used a standard, but powerful and sophisticated, software integration routine to run all our models and learning algorithms. The routine, LSODA (Petzold and Hindmarsh 1987), is in the public domain, and available over Internet.

All the algorithms have been run using a highly interactive software system in which all parameters can be varied while a model is running. The output scrolls across the screen for as long as the model runs. Thus it is possible to get a "feel" for the properties of a model, in terms of the effects of varying various parameters, that would not be possible otherwise.

When running the learning algorithms, one interesting phenomena was what we call the "6 peaks" effect: when training a network of standard neurons, the error term would eventually be periodic with period 1/6 of the target output period. Once this stage is reached, it reliably indicates that the model will reproduce the desired output signals, or in other words that the model now has the desired limit cycle.

When training the more complex models, it is important to start off with the learning rate set to zero and allow the transients in the forced equations to die out, and the system to settle into a limit cycle, before slowly turning the learning rate back on. Learning rates as high as 1.0 are quite satisfactory.

Appendix 2. Equations for a fully gap-connected network

Here we present the full set of equations for adjusting the weights and gap conductances in a network that is fully connected with ordinary synapses and has a gap junction between every pair of units. Such a net could occur as a gap-connected subset of a larger network.

Werbos P 1974 *Beyond regression: new tools for prediction and analysis in the behavioral sciences* Unpublished doctoral dissertation, Harvard University

Williams R J and Zipser D 1989 A learning algorithm for continually running fully recurrent neural networks *Neural Computation* **1** 270–80

Yamada W M, Koch C, and Adams P R 1989 Multiple channels and calcium dynamics In: *Methods in Neuronal Modeling,* eds.Koch C and Segev I (Cambridge, Mass.: MIT Press)

Jack J J B, Noble D and Tsien R W 1975 *Electric current flow in excitable cells* (Oxford: Clarendon Press)

Katz B and Miledi R 1967 The study of synaptic transmission in the absence of nerve impulses *J.Physiol.* **192** 407–436

le Cun Y 1988 A theoretical framework for back-propagation *Proc. Connectionist Models Summer School June 17–26 Carnegie–Mellon University* eds. D.Touretsky G.Hinton and T.Sejnowski 21–28

McCulloch W S and Pits W 1943 A logical calculus of the ideas immanent in neural nets *Bulletin of Mathematical Biophysics* **5** 115–37

Minsky M and Papert S 1969 *Perceptrons: an introduction to computational geometry* (Cambridge, Mass.: MIT Press)

Nagumo J S, Arimoto S, and Yoshizawa S 1962 An active pulse transmission line simulating a nerve axon *Proc. IRE* **50** 2061–70

NIPS 1988 *Neural Information Processing Systems* (American Institute of Physics: New York)

Pearlmutter B A 1989 Learning state space trajectories in recurrent neural networks *Neural Computation* **1** 263–69

Perkel D H and Mulloney B 1978 Electrotonic properties of neurons: steady state compartmental model *J. Neurophysiol.* **41** 621–39

Petzold L R and Hindmarsh A C 1987 *LSODA.. Livermore solver for ordinary differential equations, with automatic method switching for stiff and nonstiff problems* Computing and Mathematics Division, l-316, Lawrence Livermore National Laboratory, Livermore, CA 9450

Pineda F J 1987 Generalization of back-propagation to recurrent neural networks *Phys. Rev. Lett.* **59** 2229–32

Rose R M and Hindmarsh J L 1989 The assembly of currents in a thalamic neuron III. The seven-dimensional model *Proc.R.Soc.Lond.B* **237** 313–334

Rumelhart D E, Hinton G E, and Williams R J 1986 Learning representations by back-propagating errors *Nature* **323** 533–536

Rumelhart D E and McClelland J L 1986 *Parallel distributed processing* (Cambridge, Mass.: MIT Press)

Selverston A I and Moulins M 1987 *The Crustacean Stomatogastric System* (Berlin: Springer–Verlag)

References

Abbott L F and Kepler T B 1990 *Model neurons: from Hodgkin-Huxley to Hopfield* manuscript

Bekey G A 1970 System identification - an introduction and a survey *Simulation* **10** 151–166

Benson M 1979 Parameter Fitting in Dynamic Models *Ecol.Mod.* **6** 97–115

Bryson A E Jr, and Ho Y-O 1969 *Applied Optimal Control* (Blaisdell Publishing Co.)

Calabrese R L and Peterson E L 1983 Neural control of heartbeat in the leech, *Hirudo medicinalis* In: *Neural Origin of Rhythmic Movements,* A.Roberts and B.Roberts, eds. *Symp. Soc. Exp. Biol.* **37** 195–221

Connor J A and Stevens C F 1971a Voltage clamp analysis of a transient outward membrane current in gastropod neural somata *J.Physiol.* **213** 21–30

Connor J A and Stevens C F 1971b Inward and delayed outward membrane currents in isolated neural somata under voltage clamp *J.Physiol.* **213** 31–53

Doya K and Yoshizawa S 1989 Adaptive neural oscillator using continuous-time back-propagation learning *Neural Networks* **2** 375–85

Edwards D H and Mulloney B 1984 Compartmental models of electronic structure and synaptic integration in an identified neurone *J.Physiol.* **348** 89–113

Fitzhugh R 1969 Mathematical models for excitation and propagation in nerve In: *Biological Engineering,* ed. Schwan H P (McGraw Hill: New York)

Getting P 1989 Reconstruction of small neural networks In: *Methods in Neuronal Modeling,* eds.C.Koch and I.Segev (Cambridge, Mass.: MIT Press)

Graubard K, Raper J A, and Hartline D K 1938 Graded synaptic transmission between identified spiking neurons *J.Neurophysiol.* **50** 508–521

Hille B 1984 *Ionic channels of excitable membranes* (Sunderland, Mass.: Sinauer)

Hodgkin AL and Huxley A F 1952 A quantitive description of membrane current and its application to conduction and excitation in nerve *J.Physiol.* **117** 500–44

Hopfield J J 1984 Neurons with graded response have collective computational properties like those of two-state neurons *Proc.Natl.Acad.Sci.USA* **81** 3088–92

IJCNN 1990 *International Joint Conference on Neural Networks, San Diego* (IEEE and INNS)

By (3.6), the adjustment equations are:

$$\begin{aligned} \frac{dw_{ij}}{dt} &= -\eta(x_i - h_i) z^i_{W_{ij}} , \qquad \text{for } i,j = 1,2 \\ \frac{\mathrm{d}P_i}{\mathrm{d}t} &= -\eta(x_i - h_i) z^i_{P_i} , \qquad \text{for } i = 1,2 \end{aligned} \tag{9.4}$$

where $\eta > 0$ is the learning rate.

When the algorithm specified by equations 9.2, 9.3, and 9.4 are integrated, using sinusoidal (or other periodic) target outputs, the parameter values converge rapidly to a solution. In using this learning algorithm, the learning rate should be set to zero and the forced equations allowed to settle into a (forced) limit cycle, before slowly increasing the learning rate to a small positive number.

10. Summary and conclusions

We have given an exposition of a generalized, continuous-time, back-propagation algorithm. By presenting back-propagation in this way it is easy to see how to derive learning algorithms for neural network models at almost any level of sophistication. The only restriction is that the model must be presented as a set of differential equations.

We have carefully considered the interpretation of the standard network model used by connectionist modelers and suggested how it can be extended to include networks with gap junctions, to networks in which each cell has the same rest potential, and to networks in which graded synaptic transmission is the major means of communication between cells. For all these cases we have derived learning algorithms.

Our generalized approach to back-propagation algorithms also extends to include cell models with membrane currents. Thus the approach presented here for the derivation of learning algorithms will be an important tool for the analysis of realistic models of small biological neural networks.

Acknowledgments

We wish thank Gary Cottrell, Fernando Pineda, Fu-Sheng Tsung, and David Zipser for comments on the manuscript, and I-Teh Hsieh for assistance with the programming.

This research was supported by the Office of Naval Research.

The following learning algorithm can be used to train this network to produce given target outputs on each cell. Let the target outputs for the network be $\mathbf{h} = (h_1, h_2)$ and let the error term be

$$E = 1/2\{(x_1 - h_1)^2 + (x_2 - h_2)^2\}.$$

The forced equations are

$$\left.\begin{aligned}
\tau_v \frac{\mathrm{d}x_1}{\mathrm{d}t} &= -x_1 + pos(w_{11})s(h_1) + neg(w_{12})s(h_2) + P_1 q_1 \\
\tau_q \frac{\mathrm{d}q_1}{\mathrm{d}t} &= -q_1 + P_1 m_\infty(h_1) \\
\\
\tau_v \frac{\mathrm{d}x_2}{\mathrm{d}t} &= -h_2 + pos(w_{22})s(h_2) + neg(w_{21})s(h_1) + P_2 q_2 \\
\tau_q \frac{\mathrm{d}q_2}{\mathrm{d}t} &= -q_2 + P_2 m_\infty(h_2)
\end{aligned}\right\} \tag{9.2}$$

Let the sensitivity coefficients be

$$z^1_{W_{11}}, z^1_{W_{12}}, z^1_{P_1}, z^{q_1}_{P_1}$$
$$z^2_{W_{22}}, z^2_{W_{21}}, z^2_{P_2}, z^{q_2}_{P_2}$$

After transients, all other sensitivity coefficients are zero. By (3.5), the sensitivity equations are:

$$\left.\begin{aligned}
\tau_v \frac{\mathrm{d}z^1_{W_{11}}}{\mathrm{d}t} &= -z^1_{W_{11}} + pos'(W_{11})s(h_1) \\
\tau_v \frac{\mathrm{d}z^1_{W_{12}}}{\mathrm{d}t} &= -z^1_{W_{12}} + neg'(W_{12})s(h_2) \\
\tau_v \frac{\mathrm{d}z^1_{P_1}}{\mathrm{d}t} &= -z^1_{P_1} + q_1 + P_1 z^{q_1}_{P_1} \\
\tau_q \frac{\mathrm{d}z^{q_1}_{P_1}}{\mathrm{d}t} &= -z^{q_1}_{P_1} + i_\infty(h_1) \\
\\
\tau_v \frac{\mathrm{d}z^2_{W_{22}}}{\mathrm{d}t} &= -z^2_{W_{22}} + pos'(W_{22})s(h_2) \\
\tau_v \frac{\mathrm{d}z^2_{W_{21}}}{\mathrm{d}t} &= -z^2_{W_{21}} + neg'(W_{21})s(h_1) \\
\tau_v \frac{\mathrm{d}z^2_{P_2}}{\mathrm{d}t} &= -z^2_{P_2} + q_2 + P_2 z^{q_2}_{P_2} \\
\tau_q \frac{\mathrm{d}z^{q_2}_{P_2}}{\mathrm{d}t} &= -z^{q_2}_{P_2} + i_\infty(h_2)
\end{aligned}\right\} \tag{9.3}$$

derived as in section 3.

The algorithm for a network of n units, fully connected with ordinary (chemical) synapses and having a gap junction between every pair of units, is given in Appendix 2. This network has n^2 ordinary synapses and $n(n-1)/2$ gap junctions. Such a net could occur as a gap-connected subset of a larger network.

9. Algorithm for a network including membrane currents.

In this section we take a network of two neurons, each having a simplified membrane current, and show how to obtain a learning algorithm for it. The free equations are

$$\left.\begin{aligned} \tau_v \frac{\mathrm{d}x_1}{\mathrm{d}t} &= -x_1 + pos(w_{11})s(x_1) + neg(w_{12})s(x_2) + P_1 q_1 \\ \tau_q \frac{\mathrm{d}q_1}{\mathrm{d}t} &= -q_1 + P_1 m_\infty(x_1) \\ \\ \tau_v \frac{\mathrm{d}x_2}{\mathrm{d}t} &= -x_2 + pos(w_{22})s(x_2) + neg(w_{21})s(x_1) + P_2 q_2 \\ \tau_q \frac{\mathrm{d}q_2}{\mathrm{d}t} &= -q_2 + P_2 m_\infty(x_2) \end{aligned}\right\} \tag{9.1}$$

The parameters w_{11}, w_{22} are constrained to be positive and the synaptic weights w_{12}, w_{21} are constrained to be negative. The time constant τ_q is larger than the time constant τ_v. The parameters w_{11}, w_{22} model plateau potentials when sufficiently large, and the parameters w_{12}, w_{21} model inhibitory synaptic transmission. The variables q_1 and q_2 model simplified membrane currents. Each q current models a combination of a slowly activating inward current (when the membrane potential x is high) and a slowly activating outward current (when x is low). The form used for m_∞ is

$$m_\infty(x) = -tanh(x).$$

Parameters P_1, P_2, control the amplitudes of the membrane currents. Each cell can independently exhibit plateau potentials and endogenous oscillations, by setting $w_{12} = w_{21} = 0$ and choosing suitable values for the recurrent weights (w_{11} and w_{21}) and the membrane current amplitudes (P_1 and P_2).

turning off learning and forcing at this stage showing that, indeed, learning is complete (fig.6), and (fig.7) the slight drift that occurs when the free system is left to run for a prolonged period.

The detailed equations for handling a diode-like gap junction, in which current only flows in one direction, are very similar.

For larger networks with gap junctions, in which the target outputs of all units are known, it would be desirable to obtain a learning algorithm similar to the solo dancer principle for networks without gap junctions. Unfortunately an exact analogue cannot be obtained. Consider a unit i with synaptic input $w_{ij}s(x_j)$ from unit j. A change in the weight w_{ij} alters the effect of unit j on unit i but not the effect of unit i on unit j. This independence is essentially the reason why the solo dancer idea works. However a gap junction is shared by the two cells it connects, so that any change in its conductance g simultaneously affects how unit i influences unit j and how unit j influences unit i. Thus if one was to try, as in the solo dancer principle, to set up two independent learning algorithms, one for unit i and one for unit j, you have the problem that one algorithm would require the value of g to change by a certain quantity while the other algorithm requires g to change by another, different, quantity. Thus gap junction weights cannot be adjusted independently. In fact the solo dancer idea can still be applied, but now it cannot be applied to individual units i, only to **gap-connected set**s of units. The definition of a gap-connected set of units is the following. Say that unit i is **gap-connected** to unit j if there is a chain of units $i = i_1, i_2, \ldots, i_p = j$ in which there is a gap junction between each pair (i_k, i_{k+1}), for $k = 1, \ldots, p-1$. Then $G = \{i_1, \ldots, i_q\}$ is a **gap-connected set** of units if, for any unit i in G, all units k gap-connected to i are also in G. Trivially, a single unit having no gap junctions forms a gap-connected set.

The learning algorithm for a general network containing gap junctions, with target output specified for all units, can now be described. Suppose that the network of n units contains $\tilde{n}$ gap-connected subsets. The learning process consists of $\tilde{n}$ independent learning algorithms, one for each gap-connected subset. If the gap-connected subset consists of a single unit then the learning algorithm is given by equations (6.1)–(6.5). Otherwise let there be $p > 1$ units in the gap-connected subset $G_k = \{i_1, \ldots, i_p\}$. The learning algorithm adjusts all the weights and conductances of the units in G_k simultaneously, using the error term $E_k = \frac{1}{2}\sum_{j=1}^{p}\left(x_{i_j} - h_{i_j}\right)^2$, and the algorithm is

The complete learning algorithm for this network can now be given. The free equations are the equations (8.1) with g replaced by $pos(g)$. Let the target outputs for the network be $\mathbf{h} = (h_1, h_2)$ and let the error term be

$$E = 1/2\{(x_1 - h_1)^2 + (x_2 - h_2)^2\}.$$

The forced equations are

$$\begin{aligned} \tau_1 \frac{\mathrm{d}x_1}{\mathrm{d}t} + x_1 &= w_{11}\, s_{11}(h_1) + w_{12}\, s_{12}(h_2) + pos(g)(h_2 - h_1) \\ \tau_2 \frac{\mathrm{d}x_2}{\mathrm{d}t} + x_2 &= w_{21}\, s_{21}(h_1) + w_{22}\, s_{22}(h_2) + pos(g)(h_1 - h_2) \end{aligned} \tag{8.2}$$

Let the sensitivity coefficients be

$$z_{ij} = \frac{\partial x_i}{\partial w_{ij}}, \qquad \text{for } i, j = 1, 2,$$

and

$$v_i = \frac{\partial x_i}{\partial g}, \qquad \text{for } i = 1, 2.$$

Note that after transients, $\partial x_i / \partial w_{jk} = 0$ for $j \neq i$. By (3.5), the sensitivity equations are:

$$\begin{aligned} \tau_i \frac{\mathrm{d}z_{ij}}{\mathrm{d}t} &= -z_{ij} + s_{ij}(h_i), \qquad \text{for } i, j = 1, 2 \\ \tau_1 \frac{\mathrm{d}v_1}{\mathrm{d}t} &= -v_1 + pos'(g)(h_2 - h_1) \\ \tau_2 \frac{\mathrm{d}v_2}{\mathrm{d}t} &= -v_2 + pos'(g)(h_1 - h_2) \end{aligned} \tag{8.3}$$

where $pos'(g)$ is the derivative of $pos(g)$. By (3.6), the adjustment equations are:

$$\begin{aligned} \frac{\mathrm{d}w_{ij}}{\mathrm{d}t} &= -\eta(x_i - h_i) z_{ij}, \qquad \text{for } i, j = 1, 2 \\ \frac{\mathrm{d}g}{\mathrm{d}t} &= -\eta\big\{(x_1 - h_1)v_1 + (x_2 - h_2)v_2\big\}, \end{aligned} \tag{8.4}$$

where $\eta > 0$ is the learning rate.

When the equations 8.2, 8.3, and 8.4 are integrated, using sinusoidal target outputs, the parameter values converge rapidly to a solution. Figures 2 to 7 illustrate the progress of a simulation of these equations. Each of these figures is taken with only very slight editing directly from a computer screen. The stages of a simulation shown are: the beginning of learning (fig.2), the effect of turning off forcing and learning too early (fig.3), learning nearly complete (fig.4), learning complete (fig.5), the effect of

Thus after transients, $\psi_i = -1$, for $i = 1, \ldots, n$. The adjustment equation for θ requires

$$\frac{\partial E}{\partial \theta} = \sum_{i=1}^{n} (x_i - h_i)\psi_i = \sum_{i=1}^{n} h_i - \sum_{i=1}^{n} x_i.$$

From (3.6), the adjustment equations are:

$$\begin{aligned} \frac{\mathrm{d}w_{ij}}{\mathrm{d}t} &= -\eta(x_i - h_i)\phi_j, \qquad \text{for } 1 \le i, j \le n \\ \frac{\mathrm{d}\theta}{\mathrm{d}t} &= -\eta\Big\{\sum_{i=1}^{n} h_i - \sum_{i=1}^{n} x_i\Big\} \end{aligned} \tag{7.4}$$

where $\eta > 0$ is a learning rate.

In our models we generally allow the self-recurrent term w_{ii} to be non-zero. This may seem to be rather un-biological since neurons do not usually have self-recurrent synapses. However, by interpreting this term as a voltage-gated current, it is possible to provide a very simple model of plateau potentials, a phenomenon associated with many neurons in the stomatogastric ganglion of the lobster.

8. Learning algorithm for a network with gap junctions

We now consider a model and an algorithm for networks with gap junctions. Physiologically, a gap junction between two neurons consists of a channel protein, connexin, joining their membranes, which can be modelled as a conductance joining the equivalent circuits of the two cells. The free equations must include, for a gap junction with conductance g between units i and j, the current term $g(x_i - x_j)$, resp. $g(x_j - x_i)$, in the equation for unit i, resp. j. Thus for the two-unit network, the free equations are

$$\left.\begin{aligned} \tau_1 \frac{\mathrm{d}x_1}{\mathrm{d}t} &= -x_1 + w_{11}\, s(x_1) + w_{12}\, s(x_2) + g(x_2 - x_1) \\ \tau_2 \frac{\mathrm{d}x_2}{\mathrm{d}t} &= -x_2 + w_{21}\, s(x_1) + w_{22}\, s(x_2) + g(x_1 - x_2) \end{aligned}\right\} \tag{8.1}$$

We now derive a learning algorithm for this network. First of all, a preliminary adjustment must be made to (8.1). Since back-propagation will sometimes give a parameter a negative value, whereas conductances are always positive, every occurrence of a conductance g must be replaced by a restriction such as $pos(g) = e^{0.1g}$.

that the rest potential of unit i is $-\sum_{j=1}^{n} w_{ij}$, which will in general be different for every unit. This is similar to the convention in iterative network models (Rumelhart and McClelland 1986, Williams and Zipser 1989) of including an individual bias term $-\theta_i$ for each unit. The latter convention adds an extra n independent parameters to the n^2 synaptic weights of a fully connected network. However, if the live network being modeled is believed to consist of essentially identical units, then the resting potentials of all the units should be approximately the same. In that case the form of the equations should be

$$\tau_i \frac{\mathrm{d}x_i}{\mathrm{d}t} = -x_i + \sum_{j=1}^{n} w_{ij}\,\hat{s}(x_j) - \theta, \quad \text{for } 1 \le i \le n \tag{7.1}$$

where θ has the same value for all units. Consequently the learning algorithms just given in section 3 and 4 must be modified, as follows.

Assume for simplicity that all the time constants are identical so that τ can be written in place of τ_i. The free equations are the equations (7.1) above. The forced equations are:

$$\tau \frac{\mathrm{d}x_i}{\mathrm{d}t} = -x_i + \sum_{j=1}^{n} w_{ij}\,\hat{s}(h_j) - \theta, \qquad \text{for } 1 \le i \le n \tag{7.2}$$

Let the sensitivity coefficients be

$$\hat{z}^i_{jk} = \frac{\partial x_i}{\partial w_{jk}}, \qquad \text{for } 1 \le i,j,k \le n,$$

$$\psi_i = \frac{\partial x_i}{\partial \theta}, \qquad \text{for } 1 \le i \le n,$$

and let the error term be

$$E(t) = \frac{1}{2}\sum_{i=1}^{n} (x_i - h_i)^2 .$$

Because the time constants all have the same value one finds that for each j, $j = 1,\ldots,n$, the equations defining $\hat{z}^i_{ij}$, $i = 1,\ldots,n$, are all identical. So define $\phi_j = \hat{z}^1_{1j} = \hat{z}^2_{2j} = \ldots = \hat{z}^n_{nj}$, for $j = 1,\ldots,n$. As usual, for $i \neq j$, $\hat{z}^i_{jk} = 0$ after transients. Applying (3.5), the sensitivity equations are then

$$\tau \frac{\mathrm{d}\phi_j}{\mathrm{d}t} = -\phi_j + \hat{s}(h_j), \qquad \text{for } 1 \le j \le n \tag{7.3}$$

$$\text{and} \qquad \tau \frac{\mathrm{d}\psi_i}{\mathrm{d}t} = -\psi_i - 1, \qquad \text{for } 1 \le i \le n.$$

Miledi 1967) will now be described. At a synapse between presynaptic unit j and postsynaptic unit i, w_{ij} is the maximum rate of transmitter release. The transmitter crosses the synaptic cleft, binds to a receptor, and opens channels, thus causing an increase in the conductance at the postsynaptic site proportional to the amount of transmitter released. If the reversal potential for this synapse is REV_{ij}, the current entering the postsynaptic unit is

$$(x_i - REV_{ij})w_{ij}\,s(x_j) = x_i w_{ij}\,s(x_j) - REV_{ij}\,w_{ij}\,s(x_j).$$

When the first term on the right hand side is ignored we get the gross approximation referred to above, as follows. If REV_{ij} is larger and more negative than the usual operating potentials, as is often the case, then the second term on the right hand side will dominate. This can be written $+|REV_{ij}|w_{ij}\,s(x_j)$, and the combined weight $|REV_{ij}|w_{ij}$ may be replaced by the single weight w_{ij}. Thus we see that, for graded synaptic transmission, the term $w_{ij}\,s(x_j)$ is a simplistic approximation to the synaptic current in unit i caused by potential x_j in unit j.

Since synaptic characteristics vary from one synapse to the next it would be appropriate to allow a different squashing function for each synapse and write the synaptic current as $w_{ij}\,s_{ij}(x_j)$ but we have not done this yet. In effect the single output amplifier of Hopfield has been replaced by several, one at each synapse.

We have not used an explicit synaptic delay term in modeling the GM of the lobster, partly because the time scale of a synaptic delay is in the millisecond range whereas the time scale of the slow waves being modelled is $2 - 10$ seconds.

Equation (3.7) is unrealistic as it stands because it uses a symmetric squashing function $s(x)$ that ranges from -1 to $+1$. This seems to allow a synapse to change from excitatory to inhibitory, or vice versa, as x changes sign. The simplest way to handle this is to define a new non-negative squashing function $\hat{s}(x) = s(x) + 1$. Substituting in equation (3.7) gives

$$\tau_i \frac{\mathrm{d}x_i}{\mathrm{d}t} + x_i = \sum_{j=1}^{n} w_{ij}\,\hat{s}(x_j) - \sum_{j=1}^{n} w_{ij}\,.$$

Suppose for the moment that no synaptic interaction occurs between the members of this network, i.e. that the first term on the right hand side is zero. Then for each i, $1 \le i \le n$, the state variable x_i relaxes to the value $-\sum_{j=1}^{n} w_{ij}$. This effectively says

limit cycle of the whole troupe did not correspond exactly to the combination of the trained limit cycles of the individual dancers. The exact conditions for the successful use of the solo dancer principle need investigation. To be more specific, we would like to know the conditions under which the following statement is true: If each "dancer-in-training" – a periodically forced non-linear system – trains to an attracting limit cycle, then when all the dancers come together again (all the forcing inputs are replaced by the actual outputs of the trained dancers), the whole *unforced* troupe has an attracting limit cycle in which, for each i, the output of unit i closely approximates the target output used for training unit (dancer) i.

7. Neuron equivalent circuits

In this section we consider the question of the relevance of the network model (3.7) used by connectionist modellers, and by what means to extend it in the direction of more realistic models of neuron networks. In neurophysiology the electrical aspects of the functionality of a neuron - the flow of ions and the generation of electrical potentials - can be modelled by an equivalent electrical circuit.

The system (3.7) is the current equation for a neuron equivalent circuit in which the circuit potential x_i corresponds to the membrane potential of the neuron, the RC delay has a value corresponding to the membrane time constant τ_i, and the circuit has a variable current source corresponding to the synaptic current in the neuron, with time-varying strength given by the term $\sum_{j=1}^{n} w_{ij}\, s(x_j(t))$. The latter is a gross approximation to the expression used in more realistic models for the current generated by synaptic inputs (Jack, Noble, Tsien 1975). This interpretation is different from the usual one for this equation, e.g. (Hopfield 1984). Hopfield took the spike train frequency to be the output of a cell, and modelled it by an amplifier giving output $s(x_i)$ for membrane potential x_i.

In small central pattern generators, graded synaptic transmission plays a major role in pattern generation while spikes play only a minor part (Graubard et al. 1983). The main role of spikes appears to be the transmission of information out to muscles. Therefore we ignore spikes in our models and interpret the variable x_i, corresponding to the membrane potential, as the output of the cell. An interpretation of the term $w_{ij}\, s(x_j)$ in terms of the description of synaptic transmission given by (Katz and

and equation (6.5) by

$$\frac{\mathrm{d}w_{ij}}{\mathrm{d}t} = -\eta_i(s(x_i) - h_i)s'(x_i)z_{ij}$$

6.1. Derivation of the solo dancer principle

The solo dancer principle, as just described, can be derived from the equations (3.4)–(3.6). Assume network dynamics given by (3.7), and use the error term

$$E = \frac{1}{2}\sum_{k=1}^{n}(x_k - h_k)^2.$$

Using teacher forcing, the forced equations are given by (6.1) above, for all $i = 1, \ldots, n$. Let

$$\hat{z}^i_{jk} = \frac{\partial x_i}{\partial w_{jk}}, \quad \text{for } 1 \le i, j, k \le n.$$

Differentiating (6.1) with respect to w_{ij} gives

$$\tau_i \frac{\mathrm{d}\hat{z}^i_{ij}}{\mathrm{d}t} + \hat{z}^i_{ij} = s\big(h_j(t)\big),$$

which is equation (6.3) above. However, when (6.1) is differentiated with respect to w_{jl} with $j \neq i$ we get

$$\tau_i \frac{\mathrm{d}\hat{z}^i_{jl}}{\mathrm{d}t} + \hat{z}^i_{jl} = 0,$$

which means that the variables $\hat{z}^i_{jl}$, $j \neq i$, are transients which quickly become zero and can be deleted from the adjustment equations. Consequently we have

$$\frac{\partial E}{\partial w_{ij}} = \sum_{k=1}^{n} \frac{\partial x_k}{\partial w_{ij}}(x_k - h_k) = \hat{z}^i_{ij}\,(x_i - h_i), \quad \text{for } 1 \le i \le n$$

and

$$\frac{\mathrm{d}w_{ij}}{\mathrm{d}t} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta \hat{z}^i_{ij}\,(x_i - h_i) = -\eta z_{ij}(x_i - h_i)$$

which is equation (6.5) above.

The first time we used the solo dancer idea we were quite surprised that the whole network (troupe) ran perfectly after individual training was completed. So far, this is an empirical fact which has held in all our learning experiments. We were quite prepared to find that the feedback loops that appear when the whole troupe dances without any training input would cause unstable behaviour, or that the attracting

dancer interacts with all the others. The obvious way is for each dancer to learn his/her part in isolation, assuming that every other dancer performs correctly. Then when they all come together again the actual performance ought to be flawless. If each dancer achieves an attracting limit cycle in training, so that when a dancer's performance is perturbed by a small amount it quickly returns to the target, then, the performance of the troupe as a whole will reproduce the desired choreography. We call this the *solo dancer principle*.

Thus, by the solo dancer principle, each unit i can be trained in isolation. The learning process consists of n independent learning algorithms, one for each unit, that can be executed in serial or parallel. The target outputs $h_j, j = 1, \ldots, n$ are used as forcing inputs to unit i, including the case $j = i$. The forced equation for unit i, with i fixed, is:

$$\tau_i \frac{\mathrm{d}x_i}{\mathrm{d}t} + x_i = \sum_{k=1}^{n} w_{ik}\, s\big(h_k(t)\big). \tag{6.1}$$

The sensitivity coefficients for unit i are

$$z_{ij} = \partial x_i / \partial w_{ij} \quad \text{for } 1 \le j \le n \tag{6.2}$$

that satisfy the sensitivity equations

$$\left.\begin{aligned} \tau_i \frac{\mathrm{d}z_{ij}}{\mathrm{d}t} + z_{ij} &= s\big(h_j(t)\big) \\ z_{ij}(0) &= s\big(h_j(0)\big) \end{aligned}\right\} \quad \text{for } 1 \le j \le n. \tag{6.3}$$

Using the error term

$$E_i = \frac{1}{2}(x_i - h_i)^2, \tag{6.4}$$

the adjustment equation for unit i is

$$\frac{\mathrm{d}w_{ij}}{\mathrm{d}t} = -\eta_i \frac{\partial E_i}{\partial w_{ij}} = -\eta_i (x_i - h_i) z_{ij} \tag{6.5}$$

Equations 6.1, 6.3, and 6.5 are used independently for each unit. The learning rates η_i can be adjusted independently for each unit. When convergence is complete for each unit i the free equations (3.7) correctly regenerate the target behaviour $\mathbf{h}$.

If $s(x_i)$ is taken as the output of unit i, then equation (6.4) must be replaced by

$$E_i = \frac{1}{2}(s(x_i) - h_i)^2,$$

By teacher forcing, the target output h_n of unit n is used as input in place of $s(x_n)$. The forced equations are

$$\tau_i \frac{\mathrm{d}x_i}{\mathrm{d}t} + x_i = \sum_{j=1}^{n-1} w_{ij}\, s(x_j) + w_{in}\, h_n \qquad \text{for } 1 \le i \le n. \tag{5.1}$$

Define sensitivity coefficients

$$z_{pq}^i = \frac{\partial x_i}{\partial w_{pq}}, \quad 1 \le i, p, q \le n,$$

and using (3.5) the sensitivity equations are

$$\tau_i \frac{\mathrm{d}z_{pq}^i}{\mathrm{d}t} + z_{pq}^i = \sum_{j=1}^{n-1} \left\{ \delta_{ip}\, \delta_{jq}\, s(x_j) + w_{ij}\, s'(x_j) z_{pq}^j \right\} + \delta_{ip}\, \delta_{nq}\, h_n \tag{5.2}$$

where $\delta_{ij} = 1$ if $i = j$, $\delta_{ij} = 0$ otherwise. The adjustment equations are then

$$\frac{\mathrm{d}w_{pq}}{\mathrm{d}t} = -\eta \frac{\partial E}{\partial w_{pq}} = -\eta s'(x_n)\big(s(x_n(t)) - h_n(t)\big) z_{pq}^n . \tag{5.3}$$

The algorithm specified by equations (5.1), (5.2), (5.3) was derived by Doya and Yoshizawa (1989) in a different manner. In their experiments they used an approximation given by omitting the term $w_{ij}\, s'(x_j) z_{pq}^j$ on the right hand side of equation (5.2). The meaning of this approximation is that when considering the effect of another unit j on unit i, only the direct effect of unit j on unit i is taken into account; the indirect effect through other units p, q is ignored. This approximation was sufficient to solve the examples given in their paper.

6. Target output on all units: the solo dancer principle

Suppose that a network of n standard model neurons is given, a target output $h_i(t)$ is given for every unit i, $i = 1, \ldots, n$, and we want to adjust the weights so that the output of each unit approximately matches the corresponding target output. The free equations are given as usual by the system (3.7). In this case a considerable simplification is possible by an extension of the teacher forcing idea. Think of the units $1, \ldots, n$ as members of a dance troupe who have to learn their parts for a performance. Imagine the confusion if everyone tried to learn their parts simultaneously, since every

sensitivity coefficients to move the parameter vector by a small amount in the correct direction to reduce the distance between the (projection of) the state vector and the target vector.

4. Enforcing parameter restrictions

One frequently needs to ensure that the range of a parameter μ satisfies a restriction such as $\mu > 0$, $\mu < 0$, or $\alpha < \mu < \beta$, particularly when partial information is known about the parameter from biological data. This cannot be done directly since gradient descent requires that a parameter be able to take all real values. This situation can be handled as follows. Let μ be the restricted parameter and let $res(\mu^*)$ be a function, defined for all μ^*, whose range is exactly the restricted set of values. Replace each occurrence of μ in the free equations by $res(\mu^*)$ and re-derive the learning equations using 3.5 and 3.6. The result of running these learning equations will include a value for μ^* which is then converted back to a value for μ. An example will be given when we derive a learning algorithm for networks with gap junctions in section 8. Restriction functions that we have used are the following: $pos(\mu^*) = e^{\sigma\mu^*}$ with $\sigma = 0.1$, $neg(\mu^*) = -pos(\mu^*)$, $bet(\mu^*) = \alpha + (\beta - \alpha)/(1 + e^{-\sigma\mu^*})$ to enforce, respectively, $\mu > 0$, $\mu < 0$, or $\alpha < \mu < \beta$.

5. Target output on a single unit: the Doya-Yoshizawa algorithm

Here we consider a network of n units, $n > 1$, with a single oscillatory target output $h_n(t)$ specified for unit n, and the problem is to adjust the network parameters so that the network will produce a signal approximately the same as $h_n(t)$ on unit n. The standard model of a neuron is used, so the free equations for the network are given by (3.7). Let $s(x_n)$ be the actual output of unit n, and use the error term $E(t) = \frac{1}{2}(s(x_n(t)) - h_n(t))^2$. Then

$$\frac{\partial E}{\partial x_n} = s'(x_n(t))(s(x_n(t)) - h_n(t)), \text{ and } \frac{\partial E}{\partial x_i} = 0 \text{ for } i \neq n,$$

so that

$$\frac{\partial E}{\partial w_{pq}} = \frac{\partial E}{\partial x_n}\frac{\partial x_n}{\partial w_{pq}}.$$

the target vector. It is not necessary, and it will not in general be possible, for the (projection of) state vector and the target vector to be identical at all times. However, when running a system with an arbitrary set of parameter values, the projection of the state vector $\mathbf{x}(t)$ may become very distant from the target vector $\mathbf{h}(t)$. In this case the parameter changes computed by back-propagation may contradict the changes that would have been computed if the state vector was close to the target vector. The parameter vector μ may be moved out to an irrelevant area of parameter space and convergence may not occur. Pineda (personal communication) has shown that if a system does not oscillate initially then back-propagation cannot cause it do so. Teacher forcing (Williams and Zipser 1989) keeps the projection of $\mathbf{x}(t)$ close to $\mathbf{h}(t)$ by replacing, on the right hand side of equation (3.7), every occurrence of an x_i for which target output is given by its target value h_i. This ensures that the parameter changes are relevant to the task of keeping the state of the system close to the target output.

With teacher forcing the equations (3.7) become

$$\tau_i \frac{\mathrm{d}x_i}{\mathrm{d}t} + x_i = \sum_{j=1}^{n} w_{ij}\, s(y_j), \qquad \text{for } 1 \leq i \leq n \tag{3.8}$$

where $y_j = h_j$ if h_j is given, otherwise $y_j = x_j$. We introduce some terminology for the parts of a learning algorithm. The *free equations* are the original equations (3.4), for example (3.7) above. The *forced equations* are the equations obtained after modifying the free equations by teacher forcing, for example (3.8). The *sensitivity coefficients* are the variables $z_k^j = \frac{\partial x_j}{\partial \mu_k}$. (In the case of a network of standard neurons, the z_k^j variables correspond to the p_{ij}^k variables of Williams and Zipser (1989)). The *sensitivity equations* are the equations defining the sensitivity coefficients, obtained by partial differentiation from the forced equations. The exact definition of the variables z_k^j varies from one model to the next. The *adjustment equations* for adjusting parameter values are obtained from (3.6).

A generalized back-propagation algorithm can now be succinctly described using the terminology just introduced. Such an algorithm requires three sets of equations: the forced equations, the sensitivity equations, and the adjustment equations. The sensitivity equations are obtained from the forced equations by partial differentiation. The forced equations keep the state vector close to the target vector, the sensitivity equations define the sensitivity coefficients z_k^i, and the adjustment equations use the

therefore requires $n + n^3 + n^2$ equations. Note that a variable such as $z_k^i = \frac{\partial x_i}{\partial \mu_k}$ can be interpreted as follows: it measures how x_i responds to a small change in μ_k, or in other words the *sensitivity* of x_i to the parameter μ_k.

The remainder of the paper will be concerned with extensions or modifications of the following network model:

$$\tau_i \frac{\mathrm{d}x_i}{\mathrm{d}t} + x_i = \sum_{j=1}^{n} w_{ij}\, s(x_j), \qquad \text{for } 1 \leq i \leq n \tag{3.7}$$

where s is a sigmoid squashing function such as $tanh(\sigma x)$ for $\sigma > 0$. We will discuss the biological relevance of this and other network models in more detail in section 8. We note for now that the righthand side of (3.7) is interpreted as the input to a neuron, while the lefthand side acts as a delay operator, corresponding to the RC delay of the passive membrane of a neuron.

A frequently used discrete-time version of the standard network model follows from (3.7). First, all the time constants τ_i must have the same value τ. Second, take the Taylor series expansion of $x_i(t+\tau)$ at t:

$$\begin{aligned} x_i(t+\tau) &= x_i(t) + \tau x_i'(t) + O(\tau^2) \\ &= x_i(t) + \tau \cdot \frac{1}{\tau}\big(-x_i(t) + \sum_{j=1}^{n} w_{ij}\, s(x_j(t))\big) + O(\tau^2) \\ &= \sum_{j=1}^{n} w_{ij}\, s(x_j(t)) + O(\tau^2) \end{aligned}$$

The discrete-time standard model is obtained when time intervals of length τ are used and the error term of $O(\tau^2)$ is ignored:

$$x_i(t+\tau) = \sum_{j=1}^{n} w_{ij}\, s(x_j(t))$$

Back-propagation has to be augmented by a technique known as *teacher forcing* (Williams and Zipser 1989) when the target outputs are oscillatory. As just described, back-propagation computes the direction of movement in parameter space required to move, in the target subspace, the projection of the state vector $\mathbf{x}(t)$ in the direction of the target vector $\mathbf{h}(t)$. In training a network to oscillate, the essential problem is to find a set of parameter values which ensures that, if the state vector is close to the target vector in the target subspace, then the state vector will remain close to

to use an error term of the form just given. Instead, let the error term $E(t)$ be the current difference between $\mathbf{x}(t)$ and $\mathbf{h}(t)$, $F(\mathbf{x}(t), \mathbf{h}(t))$. This defines an error surface over parameter space at time t. Now take the gradient of $E(t)$ and adjust the parameter vector μ by a certain amount along the line of steepest descent. The *learning rate* η controls the size of this adjustment. Carry this out continuously, at every time t. When the parameters are being adjusted by this procedure, their values and the value of E oscillate considerably, which is to be expected since the error term used is only a coarse approximation to the true error. However, the amplitudes of the oscillations decrease to zero over the long term.

We will now give precise expression to the procedure just outlined. The gradient of $E(t)$ is the vector $\left(\frac{\partial E}{\partial \mu_1}, \ldots, \frac{\partial E}{\partial \mu_m}\right)$ in parameter space. By steepest descent, each parameter μ_k, $k = 1, \ldots, m$, is adjusted by the equation

$$\frac{\mathrm{d}\mu_k}{\mathrm{d}t} = -\eta \frac{\partial E}{\partial \mu_k}, \tag{3.1}$$

with learning rate $\eta > 0$. By the chain rule,

$$\frac{\partial E}{\partial \mu_k} = \sum_{i=1}^{n} \frac{\partial E}{\partial x_i} \frac{\partial x_i}{\partial \mu_k}, \qquad \text{for } 1 \leq k \leq m. \tag{3.2}$$

The quantities $\partial x_i / \partial \mu_k$ are obtained as follows. Partially differentiate both sides of (2.1) with respect to μ_k. Since $\mathrm{d}/\mathrm{d}t$ commutes with $\partial/\partial \mu_k$, one gets another dynamical system that defines the quantities $\partial \mathbf{x}/\partial \mu_k$, as follows:

$$\frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial \mathbf{x}}{\partial \mu_k} = \frac{\partial}{\partial \mu_k} \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \frac{\partial}{\partial \mu_k} \mathbf{f}(\mathbf{x}, \boldsymbol{\mu}) = \sum_{i=1}^{n} \frac{\partial \mathbf{f}}{\partial x_i} \frac{\partial x_i}{\partial \mu_k} \tag{3.3}$$

Now if we let $z_k^i = \frac{\partial x_i}{\partial \mu_k}$ in (3.2) and (3.3), the complete parameter adjustment algorithm consists of integrating the following three sets of equations:

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = f_i(\mathbf{x}, \boldsymbol{\mu}), \qquad \text{for } 1 \leq i \leq n \tag{3.4}$$

$$\frac{\mathrm{d}z_k^j}{\mathrm{d}t} = \sum_{i=1}^{n} \frac{\partial f_j}{\partial x_i} z_k^i \qquad \text{for } 1 \leq j \leq n, \qquad 1 \leq k \leq m \tag{3.5}$$

and

$$\frac{\mathrm{d}\mu_k}{\mathrm{d}t} = -\eta \left(\sum_{i=1}^{n} \frac{\partial E}{\partial x_i} z_k^i \right) \qquad \text{for } 1 \leq k \leq m. \tag{3.6}$$

This constitutes a dynamical system having $n + nm + m$ equations. A fully connected neural net using n standard model neurons has $m = n^2$ adjustable weights and

The number ϵ is used to decide when the output of the system is close enough to the target output for practical purposes. In general it will not be possible to reproduce the target output exactly ($\epsilon \to 0$) since the class of outputs generated by networks of the given form (2.1) may not include the desired target output. Any type of algorithm that finds a point $\hat{\mu}$ such that (a), (b), (c) hold may be said to have *converged* to the limit cycle Δ. A *learning algorithm* is a procedure that starts with a random point μ in parameter space, runs the system and compares the output with the target output, then adjusts the position of μ by a small amount. The compare-adjust cycle is repeated until the algorithm converges to a solution vector $\hat{\mu}$.

The parameter adjustment problem occurs in many areas of science and engineering (Bryson and Ho 1969, Werbos 1974). In the case of recurrent neural networks using the standard model of a neuron, the parameter adjustment problem is solved by the algorithms of Pineda (1987), Williams and Zipser (1989), and Pearlmutter (1989). In the next section we give a generalized, continuous-time formulation of back-propagation which we apply in the rest of the paper to obtain learning algorithms for various extensions and generalizations of the standard network model.

3. Back-propagation and teacher forcing

In this section we give a generalized, continuous-time, development of back-propagation for use with generalized network models.

Our development starts by considering the error term used to compare the actual output of the system and the target output. Let $F(\mathbf{x}, \mathbf{h})$ be the difference between the state vector $\mathbf{x}$ and the target output $\mathbf{h}$, measured in the target subspace G. F is usually taken to be the sum of squares $\frac{1}{2}\sum_{j=1}^{H}(x_{i_j}(t) - h_{i_j}(t))^2$, but other measures can be used. Recall that Γ is the target orbit described by $\mathbf{h}$ over time T. As mentioned in the previous section, the ideal would be to wait for the system to settle into a limit cycle Δ, then use the error function

$$E(\pi(\Delta), \Gamma) = \int_{t_0 - T}^{t_0} F(\mathbf{x}(s), \mathbf{h}(s))ds.$$

However that would require an enormous amount of computation at every step of a batch-oriented algorithm. To obtain an on-line algorithm, it does not seem possible

Suppose (2.1) is claimed to model a biological network producing steady, oscillatory, motor output. It is common in experimental situations for the motor rhythm to re-establish itself very quickly after a disturbance to the membrane potentials that does not cause physical damage. Similarly, a small change in the chemical environment of the network causes only a minor change in the ongoing rhythm. Less commonly, some small changes in the chemical environment can cause an abrupt change in the ongoing rhythm. These observations translate into mathematical conditions on the model system (2.1) as follows. The first observation says that the system's phase portrait must have an attracting limit cycle whose basin of attraction includes all areas of the state space likely to be encountered in experimental situations. We will refer to this as the *major limit cycle* of the model. The second observation says that the system is *structurally stable*, that is, the overall system behaviour remains roughly the same when the parameters are changed by a small amount. Lastly, the occurrence of an abrupt change in network output is referred to as a *bifurcation* of the system.

The given biological data is referred to as the *target output* for the system (2.1). It consists of a collection of periodic functions $\mathbf{h} = (h_{i_1}, h_{i_2}, \ldots, h_{i_H})$, $1 \leq H \leq n$, each with period $T > 0$, where for each j, $1 \leq j \leq H$, h_{i_j} specifies the desired target output of unit i_j. The *target subspace* is the subspace $G = R_{i_1} \times R_{i_2} \times \ldots \times R_{i_H}$ of the state space R^n. Let $\pi : R^n \to G$ be the projection from R^n to G. The target output $\mathbf{h}$ defines an *orbit* Γ in G with period T. In order to compare the actual output of the system (2.1) with the target output, an error function E and an $\epsilon > 0$ must be provided. Suppose that for a particular parameter vector μ, the system (2.1) has a stable limit cycle Δ. Let $\pi(\Delta)$ be the projection of Δ into the target subspace. Ideally the error function E should measure the difference, in G, between $\pi(\Delta)$ and the target orbit Γ.

The parameter adjustment problem can now be formally stated. Given a dynamical system (2.1) with parameters μ, a target output $\mathbf{h}$ with limit cycle Γ, an error function E and an $\epsilon > 0$, the problem is to find a point $\hat{\mu}$ in parameter space such that

(a) the system is structurally stable at $\hat{\mu}$;

(b) the system has a major attracting limit cycle Δ at $\hat{\mu}$;

(c) $E(\pi(\Delta), \Gamma) < \epsilon$.

we model a two-unit network, where each unit inhibits the other and each has a simplified membrane current. We show how our generalized approach enables us to derive a learning algorithm for this model.

2. Mathematical formulation of the parameter adjustment problem

The parameter adjustment problem can be loosely stated as this: some biological data is given, a model is proposed which could account for the data, and the problem is to adjust the parameters of the model to make its output match the data. The object of this section is to introduce some terminology, give the above simple idea a rigorous definition for oscillatory models, and introduce the concept of a learning algorithm as one approach to solving the parameter adjustment problem. We use the language of dynamical systems, because any continuous network model is an instance of a dynamical system, and also because this allows us to give a rigorous statement of the problem with maximum generality. This gives us a clear view of the problem and how to approach its solution.

Assume, then, that we are given a model in the form of a dynamical system

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{f}(\mathbf{x}, \mu) \tag{2.1}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ are the state variables of the system, $\mu = (\mu_1, \mu_2, \ldots, \mu_m)$ are the adjustable parameters, $\mathbf{f} = (f_1, \ldots, f_n)$ are the transfer functions that define the dynamics, and each f_i is a continuously differentiable function of $n + m$ variables. The terms *dynamical system*, *system of equations*, and *network model* are used interchangeably. The *state space* is R^n and the *parameter space* is R^m. The system (2.1) can be regarded as a network model with n units when each variable x_i is interpreted as the state of unit i, and the value of x_i is determined by the i^{th} equation of (2.1), namely $\mathrm{d}x_i/\mathrm{d}t = f_i(\mathbf{x}, \mu)$. The output of unit i can be given directly by the value of x_i or by a function $O(x_i)$. If a network model consists of p units, where the k^{th} unit is governed by a set of $n_k \geq 1$ first-order autonomous equations, then the complete set of $n = \sum_{k=1}^{p} n_k$ equations is also an instance of (2.1). Let the state variables for the k^{th} unit be $x_{k1}, x_{k2}, \ldots, x_{kn_k}$. One of these, say x_{k1}, must be designated as the output for unit k, or, in a more general formulation, the output is given by a function $O_k(x_{k1}, x_{k2}, \ldots, x_{kn_k})$ of all the state variables of the unit.

parameter adjustment. For example, if the outputs of a live network are known, but it is impossible to train the proposed model to produce these outputs, then the model must be incorrect. (This requires that the learning algorithm be complete - it must find a solution if one exists.) Another use is in the investigation of the relation between the connectivity of a network and the types of rhythm it can produce. First one trains the network to produce oscillatory output on one unit. When training is complete, the whole network oscillates and the phase relationships of all the units can be read off from the outputs. Thus the rhythms associated with particular network connectivities can be obtained.

We are particularly concerned, in understanding the biological relevance of simplified network models, to concentrate our efforts on meaningful biological models, and to bridge the gap between biologist and modeler. This paper should, therefore, appeal to two classes of readers: modelers who want to understand how to design new models and to generate new learning algorithms, and biologists who want to know the relevance of the current torrent of modeling studies (a small sample of this is provided by IJCNN (1990), NIPS (1988)) and need to know what assumptions underly some of the simpler network models used. Some sections will appeal more to the biologist, while others will appeal more to the modeler.

The paper is organized as follows. Since all continuous network models are dynamical systems (system of first order autonomous differential equations), we start out in section 2 with a rigorous mathematical formulation of the parameter adjustment problem in the context of a dynamical system. This gives us a clear view of the problem in question. In section 3 we present the backpropagation method for the generation of learning algorithms in the context of a general dynamical system. We introduce the idea of teacher forcing and some terminology for describing learning algorithms. We show in section 4 how to modify a learning algorithm to take advantage of partial knowledge about a parameter's value. In section 5 we derive the Doya-Yoshizawa algorithm and in section 6 we show that the learning algorithm takes a particularly simple form in the case when the desired output for each unit (model neuron) is given in advance. In section 7 we discuss the interpretation of simple neuron models, in particular the standard model, and suggest modifications which in turn require modified learning algorithms. In section 8 we propose a simple model for networks which include gap junctions, and a learning algorithm to go with it. In section 9

associated with gap junctions. Further, it is known that each neuron must have many different kinds of membrane currents, giving each neuron unique properties. A detailed model of such currents has been published for only a very few neurons (Connor and Stevens 1971, Yamada et al 1989), none of which are components of the model CPGs mentioned above. Many neurons of the GM CPG display one or both of the following phenomena: plateau potentials and conditional bursting capabilites. A neuron is said to have a plateau potential if a short depolarizing pulse can cause it to fire tonically for a prolonged period, and if, when firing tonically, a short hyperpolarizing pulse causes it to abruptly stop firing and return to a hyperpolarized potential. A neuron is a conditional burster if, under certain circumstances, it becomes an endogenous oscillator. That is, the neuron produces regular, periodic bursts of action potentials. There is no representation for either of these properties in the standard network model, which essentially represents each neuron as a piece of passive membrane. Thus in modelling the GM CPG we must extend the standard network model to include some representation of these phenomena, and further, we require a method of adjusting any associated parameters.

The goals of the work reported here were:

- To include gap junctions in our network models.
- To include plateau potentials and conditional bursting in the model neurons.
- To develop parameter adjustment algorithms for network models that include representations for the above properties.

We present an approach to the generation of learning algorithms for general network models. In particular we show how to obtain parameter adjustment algorithms for networks that include electrical connections, for models in which partial information is known about some parameters, and for networks in which the model neurons include membrane currents. Since plateau potentials and conditional bursting are produced by membrane currents, we are able to adjust parameters in network models that include these properties. When the actual outputs of all the units in a model network are known experimentally, which is the case for the GM CPG, we show how to obtain a learning algorithm of a particularly simple form when the standard network models is used. We indicate how to obtain the earlier recurrent BP algorithms by taking a discrete time approximation to our algorithms.

A general learning algorithm has the potential for other uses in addition to

sequence of output patterns indefinitely. Pineda (1987) extended the backpropagation (BP) algorithm to train recurrent networks to produce fixed output patterns. Williams and Zipser (1989) and Pearlmutter (1989) further extended the BP algorithm to train a recurrent network to produce time-varying output patterns. Doya and Yoshizawa (1989) showed how to train a fully recurrent network to oscillate, using a method of derivation which, superficially, was different from the method of earlier authors. Part of our work arose from an attempt to reconcile and generalize the derivation of BP algorithms. The method we developed turns out to be a generalization of the Williams-Zipser approach. The previous algorithms all used what we will call the *standard model* of a neuron. It is commonly presented as an iterative, discrete-time, algorithm in which the next state of a neuron is given by a weighted sum of its inputs, and the output of the neuron is a sigmoidal function of its current state. The output of such a unit is usually interpreted as the firing rate of a real neuron. This is a rather loose analogue of a real neuron, but is useful for obtaining suggestive insights about the possible behaviour of large networks of live neurons.

The networks we have been investigating are invertebrate central pattern generators. A central pattern generator (CPG) is a network of neurons that produces steady rhythmic output in the absence of sensory input. Such rhythmic motor patterns are ubiquitous throughout the animal kingdom. Examples include locomotion, respiration, and chewing. An important research problem in neurobiology today is to understand how CPGs function. Examples of model CPGs that have been extensively investigated include beating of leech heart (Calabrese and Peterson 1983), swimming in *Tritonia* (Getting 1989) and chewing of food in the gastric mill (GM) in the stomach of the lobster (Selverston and Moulins 1987). In the case of the GM of the lobster, all the neurons and all their interconnections involved in the production of rhythmic activity are known (Selverston and Moulins 1987).

We are studying the GM CPG of the lobster, shown in figure 1. In this, as in other CPGs, the interactions between neurons include various kinds of direct electrical connections, in addition to chemically-mediated excitatory and inhibitory synaptic transmission. The standard network model includes the latter, but does not include electrical connections, also known as *gap* junctions. If we are to build simplified models of the GM CPG we must have a representation of gap junctions in our model networks. In addition, we must have a method for adjusting the parameters

1. Introduction

A major area of interest in neurobiology is the input/output behaviour of neural networks. The corresponding network models consist of several neuron-like units, together with a model for the interactions between neurons. There is considerable variation in the level of sophistication of such models, ranging from the first abstract model of McCulloch & Pitts (1943), to the graded response model of Hopfield(1984), to Hodgkin-Huxley type models (Hodgkin and Huxley 1952, Rose and Hindmarsh 1989, Yamada et al. 1989) and to multi-compartment models of a single neuron (Perkel and Mulloney 1978, Edwards and Mulloney 1984). The adjustment of parameters that cannot be measured experimentally is an important problem in the development of network models. If there are many parameters with unknown values then the problem is difficult because in general all the parameters may interact with each other in complex ways and adjusting one parameter may affect the response of the model to other parameters. One would like to have a way to adjust parameter values systematically. This is the problem addressed by learning algorithms. A learning algorithm is NOT about learning in the biological sense, but in the sense of automatically adjusting a complex interacting group of biological parameters. Nonetheless this does not preclude the possibility that some future learning algorithm may be a useful model of biological learning.

Backpropagation was first introduced to the neural network community by Rumelhart, Hinton and Williams (1986). They showed how to train a multi-layer, feed-forward, network of neuron-like units to produce a desired set of static input-output relationships. This was the first major advance on the problem of learning in neural nets since the perceptron convergence theorem of Minsky and Papert (1969). Although this was the first appearance of backpropagation in the neural network literature, it appears in the 1960s in a control theory textbook (Bryson and Ho 1969) and survey paper (Bekey 1970), and in the 1970s in economics (Werbos 1974). In a multi-layer network, information flows in only one direction, from input units to output units, layer by layer. Such a network cannot produce time-varying outputs because once the input information has reached the output units no more change is possible. In a recurrent network, information can flow in both directions between any two units. Such a network may settle into a fixed output pattern or may continue to cycle through a

Contents

		Page
1.	Introduction	3
2.	Formulation of the parameter adjustment problem	7
3.	Backpropagation and teacher forcing	9
4.	Enforcing parameter restrictions	13
5.	Target output on a single unit: Doya-Yoshizawa algorithm	13
6.	Target output on all units: the solo dancer principle	14
7.	Neuron equivalent circuits	17
8.	Algorithm for networks with gap junctions	20
9.	Algorithm for a network including a membrane current	22
10.	Summary and conclusions	25
	Acknowledgements	xx
	References	xx
	Appendix 1	xx
	Appendix 2	xx

Abstract.

One of the most important problems for studying neural network models is the adjustment of parameters. Here we show how to formulate the problem as the minimization of the difference between two limit cycles. The backpropagation (BP) method for learning algorithms is described as the application of gradient descent to an error function that computes this difference. A mathematical formulation is given that is applicable to any type of network model, and applied to several models. For example, when learning in a network in which all cells have a common, adjustable, bias current, the value of the bias is adjusted at a rate proportional to the difference between the sum of the target outputs and the sum of the actual outputs. When learning in a network of n cells where a target output is given for every cell, the learning algorithm splits into n independent learning algorithms, one per cell. For networks containing gap junctions, a gap junction is modelled as a conductance times the potential difference between the two adjacent cells. The requirement that a conductance g must be positive is enforced by replacing g by a function $pos(g^*)$ whose value is always positive, for example $exp(0.1g^*)$, and deriving an algorithm that adjusts the parameter g^* in place of g. When target output is specified for every cell in a network with gap junctions, the learning algorithm splits into fewer independent components, one for each gap-connected subset of the network. The learning algorithm for a gap-connected set of cells cannot be parallelized further. As a final example, a learning algorithm is derived for a mutually inhibitory two-cell network in which each cell has a membrane current.

This generalized approach to backpropagation allows one to derive a learning algorithm for almost any model neural network given in terms of differential equations. It will be an essential tool for adjusting parameters in small but complex network models.

Learning algorithms for oscillatory networks with gap junctions and membrane currents

Peter F Rowat and Allen I Selverston

Biology Department 0322, University of California at San Diego, La Jolla, California 92093-0322, U.S.A

Short title: *Learning in oscillatory networks*

To be published in:

NETWORK: Computation in Neural Systems

Volume 2 Issue 1 February 1991.