Appendix B

Chain-rule propagation algorithms

B.1 Introduction[1]

Adaptive dynamic systems (dynamic systems which learn to change over time) may be implemented using neural networks as the adaptive component. Such systems include adaptive process controllers [86], adaptive filters, and multi-layer networks with output feedback connections [130]. This appendix outlines some of the "traditional" gradient based training approaches for these systems, to contrast with the other methods presented in this thesis.

In the neural network literature, training algorithms for such systems are generally of two types: those which propagate derivative information forwards in time, and those which propagate it backwards. These two types of algorithm are derived and analyzed for a simple prototype system. It is shown that they are very closely related because they compute the same components of the gradient vector but in a different order. The well known computational properties of each algorithm are then explained using a simple matrix multiplication analogy. Extensions of the prototype to control systems are demonstrated.

A discrete-time prototype of such systems is shown in figure B.1. At time t in this figure, F_t is the "system function" (which incorporates the adaptive component), the vector $\mathbf{y}_t$ contains the system state variables, and the scalar c_t is the time step cost. The vector $\mathbf{w}$ contains the parameters (or "weights") that are fed to the adaptive components of all F_t (the vectors $\mathbf{w}_t$ are all equal to $\mathbf{w}$). The goal of training

[1] This appendix is derived from the author's paper in the Proceedings of the 1995 ANNES conference [114].

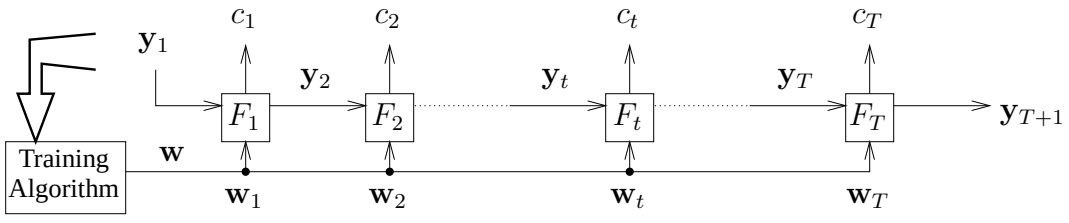

Figure B.1: *A generic discrete-time adaptive dynamic system.*

(or adaption) is to adjust the weights $\mathbf{w}$ to minimize

$$E = \sum_{i=1}^{T} c_i \tag{B.1}$$

This prototype can represent many different learning problems by selecting the F_t appropriately. For example, F_t could represent a process and its controller at time t, so that by minimizing E we ensure optimal process control. This is elaborated further in section B.4.

To train such systems using gradient based methods[2] we must calculate the gradient of the weight vector with respect to E. Two commonly cited algorithms for computing this are backpropagation-through-time (BPTT) and real-time-backpropagation (RTBP). BPTT [96, 95] finds the system states forward in time and then propagates information backwards through time to find the gradient. RTBP [130, 99] allows the gradient to be computed forward in time as the system states become known. RTBP is known to be more computationally expensive than BPTT but it does not require a backwards-though-time phase, so it can be performed on-line[3].

These two algorithms are equivalent in terms of what they compute, but because few authors derive both for the same system (and because notation differs between authors) it is not always obvious that they are interchangeable.

This appendix shows how both types of learning algorithm can be derived for the prototype system in figure B.1. In both derivations, application of the chain rule shows that information must be propagated between adjacent time steps for the gradient to be computed. This can be done either forwards or backwards in time. Thus these algorithms are named forward propagation (FP) and backward propagation (BP).

Concepts similar to [12] are presented below, though in that paper RTBP and BPTT were related using the inter-reciprocity of signal flow graphs. Here the application of matrix chain-rule techniques are emphasized.

B.2 Derivation of FP and BP

B.2.1 Notation

Notational standards for writing vector derivatives vary widely between authors, so we must define our own here. We define the derivative of vector $\mathbf{a}$ (of size n_a) with respect to vector $\mathbf{b}$ (of size n_b) as the matrix $\mathrm{d}\mathbf{a}/\mathrm{d}\mathbf{b}$ (of size $n_a \times n_b$), whose (i,j)th element is $\mathrm{d}a_i/\mathrm{d}b_j$ (a_i and b_i are elements of $\mathbf{a}$ and $\mathbf{b}$). The partial derivative matrix $\partial\mathbf{a}/\partial\mathbf{b}$ is defined similarly. Note the following convention for partial derivatives: $\partial\mathbf{a}/\partial\mathbf{b}$ is calculated assuming that only $\mathbf{b}$ varies while all other quantities in the *definition* of $\mathbf{a}$ are held constant. The total derivative $\mathrm{d}\mathbf{a}/\mathrm{d}\mathbf{b}$ assumes that *all* of $\mathbf{b}$'s influences on $\mathbf{a}$ are accounted for. For example, if

$$\mathbf{a} \triangleq F_1(\mathbf{b}, \mathbf{c}), \quad \mathbf{c} \triangleq F_2(\mathbf{b}), \qquad \left(\begin{array}{c} \mathbf{b} \\ \rightarrow \boxed{F_1} \rightarrow \mathbf{a} \\ \rightarrow \boxed{F_2} \rightarrow \mathbf{c} \end{array} \right) \tag{B.2}$$

[2] Of which gradient descent is the simplest.

[3] An on-line training algorithm is one in which the training process occurs during the operation of the system. On-line algorithms tend to be simpler to implement.

then

$$\frac{\mathrm{d}\,\mathbf{a}}{\mathrm{d}\,\mathbf{b}} = \frac{\partial\,\mathbf{a}}{\partial\,\mathbf{b}} + \frac{\partial\,\mathbf{a}}{\partial\,\mathbf{c}} \cdot \frac{\mathrm{d}\,\mathbf{c}}{\mathrm{d}\,\mathbf{b}} \tag{B.3}$$

It can be easily shown that the chain rule holds in the vectorial case, with one *caveat*: it must be remembered that matrix derivative quantities are not commutative, so that

$$\frac{\partial\,\mathbf{a}}{\partial\,\mathbf{c}} \cdot \frac{\mathrm{d}\,\mathbf{c}}{\mathrm{d}\,\mathbf{b}} \neq \frac{\mathrm{d}\,\mathbf{c}}{\mathrm{d}\,\mathbf{b}} \cdot \frac{\partial\,\mathbf{a}}{\partial\,\mathbf{c}} \tag{B.4}$$

B.2.2 Forward propagation algorithm (FP)

The FP algorithm is now derived. Using the chain rule and the definition of E we get:

$$\frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{w}} = \sum_{i=1}^{T} \frac{\partial\,E}{\partial\,c_i} \cdot \frac{\mathrm{d}\,c_i}{\mathrm{d}\,\mathbf{w}} \tag{B.5}$$

$$= \sum_{i=1}^{T} \left(\frac{\partial\,c_i}{\partial\,\mathbf{w}} + \frac{\partial\,c_i}{\partial\,\mathbf{y}_i} \cdot \frac{\mathrm{d}\,\mathbf{y}_i}{\mathrm{d}\,\mathbf{w}} \right) \tag{B.6}$$

(Note that $\partial E/\partial c_i = 1$). If the values of $\mathbf{y}_i$ and $\mathbf{w}$ are known (i.e. the system has reached at least time step i) then the quantities $\partial c_i/\partial \mathbf{w}$ and $\partial c_i/\partial \mathbf{y}_i$ can be computed without any further information — assuming of course that our knowledge of the function F_i is sufficient. We can find $\mathrm{d}\mathbf{y}_i/\mathrm{d}\mathbf{w}$ if we know the previous $\mathrm{d}\mathbf{y}_{i-1}/\mathrm{d}\mathbf{w}$, using the chain rule:

$$\frac{\mathrm{d}\,\mathbf{y}_i}{\mathrm{d}\,\mathbf{w}} = \frac{\partial\,\mathbf{y}_i}{\partial\,\mathbf{w}} + \frac{\partial\,\mathbf{y}_i}{\partial\,\mathbf{y}_{i-1}} \cdot \frac{\mathrm{d}\,\mathbf{y}_{i-1}}{\mathrm{d}\,\mathbf{w}} \tag{B.7}$$

Note that $\mathrm{d}\mathbf{y}_1/\mathrm{d}\mathbf{w} = 0$. Thus the FP algorithm is:

- Set $\mathrm{d}\mathbf{y}_1/\mathrm{d}\mathbf{w} = 0$ and $\mathrm{d}E/\mathrm{d}\mathbf{w} = 0$
- For $t = 1 \ldots T$ do
  - Calculate c_t and $\mathbf{y}_{t+1}$ using F_t (or measure these things, depending on the system).
  - Calculate $\mathrm{d}c_i/\mathrm{d}\mathbf{w}$ and add it to $\mathrm{d}E/\mathrm{d}\mathbf{w}$ (using equations B.5–B.6).
  - Calculate $\mathrm{d}\mathbf{y}_{t+1}/\mathrm{d}\mathbf{w}$ from the value of $\mathrm{d}\mathbf{y}_t/\mathrm{d}\mathbf{w}$ (using equation B.7).

B.2.3 Backward propagation algorithm (BP)

The BP algorithm is now derived. This time $\mathrm{d}E/\mathrm{d}\mathbf{w}$ is split up into different components from the FP approach:

$$\frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{w}} = \sum_{i=1}^{T} \frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{w}_i} \tag{B.8}$$

$$= \sum_{i=1}^{T} \left(\frac{\partial\,E}{\partial\,c_i} \cdot \frac{\partial\,c_i}{\partial\,\mathbf{w}} + \frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{y}_{i+1}} \cdot \frac{\partial\,\mathbf{y}_{i+1}}{\partial\,\mathbf{w}} \right) \tag{B.9}$$

$$= \sum_{i=1}^{T} \left(\frac{\partial\,c_i}{\partial\,\mathbf{w}} + \frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{y}_{i+1}} \cdot \frac{\partial\,\mathbf{y}_{i+1}}{\partial\,\mathbf{w}} \right) \tag{B.10}$$

Method	*Time taken*	*Storage required*
FP	$\mathcal{O}\Big(T\, n_y^2\, n_w\Big)$	$\mathcal{O}\Big(n_y\, n_w\Big)$
BP	$\mathcal{O}\Big(T(n_y^2 + n_y\, n_w)\Big)$	$\mathcal{O}\Big(T\, n_y\Big)$

Table B.1: *Efficiency of the FP and BP algorithms.*

Note that the derivative $\mathrm{d}E/\mathrm{d}\mathbf{w}_i$ assumes that only the weight vector into function F_i is varied while all others are held constant. The quantity $\mathrm{d}E/\mathrm{d}\mathbf{y}_i$ can be calculated from the subsequent $\mathrm{d}E/\mathrm{d}\mathbf{y}_{i+1}$, because if $\mathbf{y}_i$ is varied then only subsequent $\mathbf{y}$ and c values will change. Thus, using the chain rule again:

$$\frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{y}_i} = \frac{\partial\, c_i}{\partial\, \mathbf{y}_i} + \frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{y}_{i+1}} \cdot \frac{\partial\, \mathbf{y}_{i+1}}{\partial\, \mathbf{y}_i} \tag{B.11}$$

Note that $\mathrm{d}E/\mathrm{d}\mathbf{y}_{T+1} = 0$. Thus the BP algorithm is:

- For $t = 1 \ldots T$ do
  - Calculate (or measure) $\mathbf{y}_{t+1}$ and c_t, using F_t.
- Set $\mathrm{d}E/\mathrm{d}\mathbf{w} = 0$ and $\mathrm{d}E/\mathrm{d}\mathbf{y}_{T+1} = 0$
- For $t = T \ldots 1$ do
  - Calculate $\mathrm{d}E/\mathrm{d}\mathbf{y}_t$ from $\mathrm{d}E/\mathrm{d}\mathbf{y}_{t+1}$ (using equation B.11).
  - Calculate $\mathrm{d}E/\mathrm{d}\mathbf{w}_t$ and add it to $\mathrm{d}E/\mathrm{d}\mathbf{w}$ (using equations B.8–B.10).

B.3 Efficiency of FP and BP

B.3.1 Space and time requirements

Table B.1 shows the order[4] of the time and storage space requirements for the two algorithms. Here n_y and n_w are the sizes of the $\mathbf{y}_t$ and $\mathbf{w}$ vectors respectively. The time values were determined assuming that the "local" partial derivatives of each system function F_t (i.e. $\partial\mathbf{y}_{t+1}/\partial\mathbf{y}_t$, $\partial\mathbf{y}_{t+1}/\partial\mathbf{w}$, $\partial c_t/\partial\mathbf{y}_t$, $\partial c_t/\partial\mathbf{w}$) are relatively fast to compute. Thus in both cases the time taken is influenced most by the matrix multiplications required by the chain rule.

The storage space values correspond to the amount of information which needs to be recorded about the system for later use. In the FP algorithm this is the matrix $\mathrm{d}\mathbf{y}_t/\mathrm{d}\mathbf{w}$, in the BP algorithm this is the system state $\mathbf{y}_t$ stored over all time.

FP is slower than BP by an approximate factor of

$$\eta_{\text{time}} \approx \frac{1}{n_w} + \frac{1}{n_y} \tag{B.12}$$

For a large system[5] this means that FP can be significantly slower than BP. The storage space required by FP is less than that of BP by a factor of

$$\eta_{\text{space}} \approx \frac{T}{n_w} \tag{B.13}$$

[4]The "Big-Oh" notation used in this table just shows the dominant terms in the time and space expressions, for large systems.
[5]i.e. where n_y and n_w are large

When $T \gg n_w$ (for example when the system is being run for a long time with a small time step) FP uses much less storage space than BP.

The advantage of BP is that it is fast. The advantages of FP are that it can use less storage space than BP, and it can be implemented as an on-line algorithm (where learning occurs forward in time as the system operates).

B.3.2 Why is FP slower than BP?

It has been shown above that FP is slower than BP, but the derivations of the algorithms do not make the fundamental reason for this very clear.

Consider the simplified situation[6] where all the $c_i = 0$ except for c_T. A closed form expression for $\mathrm{d}E/\mathrm{d}\mathbf{w}$ can be found by "unfolding" the BP algorithm over time. The result can be expressed in terms of the local partial derivatives of each system block F_t, as follows:

$$\begin{aligned} \frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{y}_i} &= \frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{y}_{i+1}}\,\mathbf{Q}_i && \text{(B.14)} \\ &= \frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{y}_T}\Big[\mathbf{Q}_{T-1}\mathbf{Q}_{T-2}\cdots\mathbf{Q}_{i+1}\mathbf{Q}_i\Big] && \text{(B.15)} \\ &= \frac{\partial\,c_T}{\partial\,\mathbf{y}_T}\Big[\mathbf{Q}_{T-1}\mathbf{Q}_{T-2}\cdots\mathbf{Q}_{i+1}\mathbf{Q}_i\Big] && \text{(B.16)} \end{aligned}$$

where $\mathbf{Q}_i = \partial\mathbf{y}_{i+1}/\partial\mathbf{y}_i$, and

$$\begin{aligned} \frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{w}} &= \frac{\partial\,c_T}{\partial\,\mathbf{w}} + \sum_{i=1}^{T}\frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{y}_{i+1}}\,\mathbf{P}_{i+1} && \text{(B.17)} \\ &= \frac{\partial\,c_T}{\partial\,\mathbf{w}} + \frac{\partial\,c_T}{\partial\,\mathbf{y}_T}\Big[\mathbf{Q}_{T-1}\cdots\mathbf{Q}_3\mathbf{Q}_2\mathbf{P}_2 + \mathbf{Q}_{T-1}\cdots\mathbf{Q}_4\mathbf{Q}_3\mathbf{P}_3 + \cdots\cdots \\ &\qquad +\mathbf{Q}_{T-1}\mathbf{P}_{T-1} + \mathbf{P}_T\Big] && \text{(B.18)} \end{aligned}$$

where $\mathbf{P}_i = \partial\mathbf{y}_i/\partial\mathbf{w}$. The FP algorithm can also be unfolded in this way to yield the same expression. The difference between the two algorithms is the order in which the terms of the form

$$\frac{\partial\,c_T}{\partial\,\mathbf{y}_T}\mathbf{Q}_{T-1}\cdots\mathbf{Q}_{i+1}\mathbf{Q}_i\mathbf{P}_i \qquad \text{(B.19)}$$

are constructed. In the FP algorithm these terms are built up right-to-left because the values earliest in time are on the right. Thus the intermediate products have the form

$$\mathbf{Q}_k\mathbf{Q}_{k-1}\cdots\mathbf{Q}_{i+1}\mathbf{Q}_i\mathbf{P}_i \qquad (k > i) \qquad \text{(B.20)}$$

where k is the "iteration step". This quantity is a $n_y \times n_w$ matrix, so pre-multiplying it by $\mathbf{Q}_{k+1}$ requires $\mathcal{O}(n_y^2\,n_w)$ time. In the BP algorithm these terms are built up left-to-right, so the intermediate products have the form

$$\frac{\partial\,c_T}{\partial\,\mathbf{y}_T}\mathbf{Q}_{T-1}\cdots\mathbf{Q}_{k+1}\mathbf{Q}_k \qquad (k < T) \qquad \text{(B.21)}$$

[6] This simplification is made to keep the equations manageable—it does not affect the result.

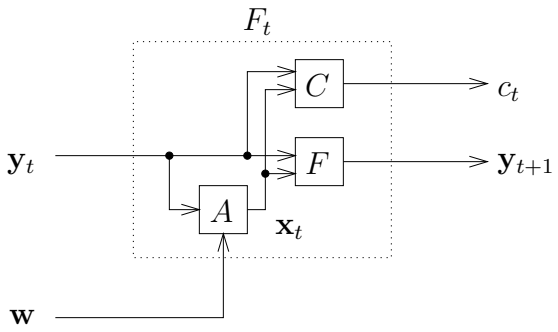

Figure B.2: *A system function for a stateless controller.*

FP

$$\frac{\mathrm{d}\,\mathbf{y}_{i+1}}{\mathrm{d}\,\mathbf{w}} = \left(\frac{\partial\,\mathbf{y}_{i+1}}{\partial\,\mathbf{y}_i} + \frac{\partial\,\mathbf{y}_{i+1}}{\partial\,\mathbf{x}_i}\cdot\frac{\partial\,\mathbf{x}_i}{\partial\,\mathbf{y}_i}\right)\cdot\frac{\mathrm{d}\,\mathbf{y}_i}{\mathrm{d}\,\mathbf{w}} + \frac{\partial\,\mathbf{y}_{i+1}}{\partial\,\mathbf{x}_i}\cdot\frac{\partial\,\mathbf{x}_i}{\partial\,\mathbf{w}}$$

$$\frac{\mathrm{d}\,c_i}{\mathrm{d}\,\mathbf{w}} = \left(\frac{\partial\,c_i}{\partial\,\mathbf{x}_i}\cdot\frac{\partial\,\mathbf{x}_i}{\partial\,\mathbf{y}_i} + \frac{\partial\,c_i}{\partial\,\mathbf{y}_i}\right)\frac{\mathrm{d}\,\mathbf{y}_i}{\mathrm{d}\,\mathbf{w}} + \frac{\partial\,c_i}{\partial\,\mathbf{x}_i}\cdot\frac{\partial\,\mathbf{x}_i}{\partial\,\mathbf{w}}$$

BP

$$\frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{y}_i} = \frac{\partial\,c_i}{\partial\,\mathbf{y}_i} + \frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{y}_{i+1}}\left(\frac{\mathrm{d}\,\mathbf{y}_{i+1}}{\mathrm{d}\,\mathbf{y}_i} + \frac{\partial\,\mathbf{y}_{i+1}}{\partial\,\mathbf{x}_i}\cdot\frac{\partial\,\mathbf{x}_i}{\partial\,\mathbf{y}_i}\right)$$

$$\frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{w}_i} = \left(\frac{\partial\,c_i}{\partial\,\mathbf{x}_i} + \frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{y}_{i+1}}\cdot\frac{\partial\,\mathbf{y}_{i+1}}{\partial\,\mathbf{x}_i}\right)\frac{\partial\,\mathbf{x}_i}{\partial\,\mathbf{w}}$$

Table B.2: *FP and BP propagation equations for a control system.*

This quantity is a $1 \times n_y$ matrix, so post-multiplying it by $\mathbf{Q}_{k-1}$ requires $\mathcal{O}(n_y^2)$ time, and the final post-multiplication by $\mathbf{P}_i$ requires $\mathcal{O}(n_y\, n_w)$ time.

Thus FP is slower than BP because a much larger amount of information needs to be propagated from step to step (FP and BP propagate $n_y \times n_w$ and $1 \times n_y$ matrices respectively between time steps). This happens (in the above example) because in FP the value of $\partial c_T/\partial \mathbf{y}_T$ is not known until the end of the training process and so a large amount of information must be carried through time to allow for its unknown value.

B.4 Control system extension to the prototype

To solve a particular learning problem, the functions F_t in the prototype must be specified in more detail. Figure B.2 shows an example of this for a stateless controller. Here F is the discrete transfer function of the process or plant, and A is the adaptive controller which produces the control output $\mathbf{x}_t$. A is adapted via the weight vector $\mathbf{w}$. The cost function C can be designed so that the minimum total cost corresponds to optimal system control in some sense.

It is relatively easy to derive the propagation equations to use in the FP and BP algorithms. For

reference these equations are shown in table B.2. In these equations, terms of the form

$$\frac{\partial\, k}{\partial\, \mathbf{x}_i} \cdot \frac{\partial\, \mathbf{x}_i}{\partial\, \mathbf{y}_i} \qquad \text{and} \qquad \frac{\partial\, k}{\partial\, \mathbf{x}_i} \cdot \frac{\partial\, \mathbf{x}_i}{\partial\, \mathbf{w}}$$

(where k is some vector or scalar) imply backpropagation operations in the controller A, i.e. we are trying to find an input gradient ($\partial k/\partial \mathbf{y}_i$ or $\partial k/\partial \mathbf{w}$) given an output gradient ($\partial k/\partial \mathbf{x}_i$). If A is a feed-forward neural network (such as a multi-layer perceptron) then these backpropagation operations can be computed using the standard neural network backpropagation equations. Note that doing this is usually faster (and uses less storage) than computing all the elements of $\partial \mathbf{x}_i/\partial \mathbf{y}_i$ or $\partial \mathbf{x}_i/\partial \mathbf{w}$ and performing the matrix multiplication.